Causes of the Violation of Integrity Constraints for Supporting the Quality of Databases

Conference Paper · June 2011

DOI: 10.1007/978-3-642-21934-4_24 · Source: DBLP

Hendrik Decker
Institut für Informatik

Causes of the Violation of Integrity Constraints for Supporting the Quality of Databases

Hendrik Decker

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Causes of the Violation of Integrity Constraints for Supporting the Quality of Databases

Hendrik Decker [⋆]

Instituto Tecnológico de Informática, Valencia, Spain

Abstract. The quality of the information provided by databases can be captured by integrity constraints. Thus, violated cases of constraints may serve as a basis for measuring the quality of given database states. A quality metric with the potential of more accuracy is obtained by measuring the causes, i.e., data that are responsible for constraint violations. Such measures also serve for controlling quality impairment across updates.

1 Introduction

In [5, 7], we have shown that the quality of stored data can be modeled, measured and monitored by declarative integrity constraints. The latter describe semantic properties of stored data that are required to hold in each database state. Thus, violations of such constraints reflect a lack of data quality. Hence, as soon as violations of integrity can be quantified, the quality of the data on which the constraints are imposed can be measured. Quality can then also be controlled, by checking, upon each update, whether the amount of integrity violation increases or decreases. We are going to see that most, though not all known integrity checking methods in the literature can be used for this kind of control.

Constraint violations can be quantified in several ways. In [5, 7], the basic idea of quantifying the quality of stored data was to count the number of violated cases, i.e. instances of constraints that are not satisfied. Not only the cardinality of sets of violated cases, but also these sets themselves constitute a valuable metric space, together with the partial order of set inclusion.

In this paper, we show that, instead of *cases*, also the *causes* of violated cases can be counted, for measuring the amount of extant inconsistency in a database. And, similar to cases, also the powerset of causes of violations constitutes a metric space in which sets of causes can be compared for monitoring whether given updates would introduce new integrity violations or not. Causes essentially are minimal sets of stored data that are sufficient for violating some constraint. They are easily determined for denial constraints without negated literals, but more involved if there is non-monotonic negation.

While cases of violated constraints are only statements *about* the lack of quality of stored data, causes precisely *are* the data that lack quality. Hence, counting

⋆ partially supported by FEDER and the Spanish grants TIN2009-14460-C03 and TIN2010-17139

causes and checking their increase or decrease may provide more accurate quality metrics than cases. Essentially, the main idea of this paper is to improve the original concept of quantifying the quality of databases as elaborated in [5, 7], by replacing cases with causes in the main definitions.

In Section 2, we outline the formal framework of this paper. In Section 3, we revisit the definition of cases, originally used for inconsistencytolerant integrity checking [6]. Cases are instances of constraints that are apt to model the quality of information in a more differentiated manner than universally quantified constraints. In Section 4, we discuss several inconsistency metrics. In particular, we revisit the metrics based on cases, as introduced in [5, 7], and introduce metrics based on causes in an analogous manner. In Section 5, we show that, apart from providing quality metrics, measures of integrity violations based on cases or causes also may serve to control and contain impaired data quality across updates. In Section 6, an outlook to future work concludes the paper.

2 Background and Framework

Background and framework of this paper, including notations and terminology, are consititued by *datalog*, as referred to in the standard literature, e.g., [1] [9].

2.1 Databases, Updates, Constraints

An *atom* is an expression of the form $p(a_1, ..., a_n)$, where p is a predicate of arity n ($n \geq 0$); the a_i, called *arguments*, are either constants or variables. A *literal* is either an atom A or a negated atom $\sim A$. A *fact* is an atom where all arguments are constants. A *database clause* is either a *fact*, the predicate of which corresponds to a relational table and the arguments of which correspond to column values in that table, or a formula of the form $A \leftarrow B$, where the *head* A is an atom and the *body* B is a conjunction of literals; all variables in $A \leftarrow B$ are implicitly quantified universally in front of the formula.

A *database* is a finite set of database clauses. For each database D and each predicate p, the set of clauses D_p in D the head of which has predicate p is called the *definition* of p in D. If q is a predicate in the body of some clause in D_p, then p is said to *recur* on q; p is also said to *recur* on each predicate on which q recurs. A database D is *definite* if there is no negated atom in the body of any clause in D. It is *hierarchical* if there is no predicate in D that recurs on itself.

An *update* is a finite set of database clauses to be inserted or deleted. For an update U of a database state D, we denote the updated database, where all inserts in U are added to D and all deletes in U are removed from D, by D^U.

An *integrity constraint* (in short, *constraint*) is a first-order predicate logic sentence which, for convenience and without loss of expressive power, we assume to be always represented as *denials*, i.e., formulas of the form $\leftarrow B$, where the body B states what must not hold. Implicitly, each variable in B is universally quantified at the front of $\leftarrow B$. A denial is *definite* if there is no negated atom in its body. An *integrity theory* is a finite set of constraints.

Constraints can be read as *necessary* conditions for the quality of stored data. If all intended application semantics are expressed by constraints, then the integrity theory represents a complete set of conditions that, when satisfied, is also *sufficient* for ensuring the quality of the stored data.

The DBMS is supposed to ensure that the database satisfies its integrity theory at all times, i.e., that all constraints are logical consequences of each state. To achieve this, database theory requires that, for each update U, the 'old' database D, i.e., the state to be updated by U, must satisfy all constraints, such that integrity checking can focus on those constraints that are possibly affected by the update. If those constraints remain satisfied, then the 'new' state D^U reached by committing U also satisfies all constraints.

Despite the theoretical insistence on total consistency at all times, the quality of a database is likely to suffer and deteriorate in the course of its evolution. Hence, it is necessary to have mechanisms that are able to tolerate certain amounts of integrity violations. As we are going to see, the cause-based approach developed in this paper is inconsistency-tolerant.

From now on, let D, IC, I, U and adornments thereof always stand for a database, an integrity theory, a constraint and, resp., an update. For convenience, we write $D(I) = true$ (resp., $D(I) = false$) if I is satisfied (resp., violated) in D. Similarly, $D(IC) = true$ (resp., $D(IC) = false$) means that all constraints in IC are satisfied in D (resp., at least one constraint in IC is violated in D).

2.2 Integrity Checking

The quality of stored data that is described by integrity constraints can be controlled by checking integrity for each update that could potentially violate it.

Below, we revisit a definition of integrity checking that abstracts away from any technical detail of how checking is done [6]. It describes each integrity checking method $\mathcal{M}$ as a mapping that takes as input a database D, and integrity theory IC and an update U, and outputs either *sat* or *vio*. If $\mathcal{M}$ is sound, $\mathcal{M}(D, IC, U) = sat$ indicates that U preserves integrity satisfaction, i.e., $D^U(IC) = true$. If $\mathcal{M}$ is also complete, then $\mathcal{M}(D, IC, U) = vio$ indicates that U violates integrity i.e., $D^U(IC) = false$. Also the output *vio* of an incomplete method may mean that the update would violate integrity. But it may also mean that further checking is needed for determining the integrity status of D^U; if there are not enough resources to do so, then U should be cautiously rejected.

Definition 1. (*Sound and complete integrity checking*)
Let $\mathcal{M}$ be a method for integrity checking. $\mathcal{M}$ is called *sound* or, resp., *complete* if, for each (D,IC,U) such that $D(IC)$=*true,* (1) or, resp., (2) holds.

$$\text{If } \mathcal{M}(D, IC, U) = sat \text{ then } D^U(IC) = true. \tag{1}$$

$$\text{If } D^U(IC) = true \text{ then } \mathcal{M}(D, IC, U) = sat. \tag{2}$$

Quality maintenance by integrity checking would tend to be too expensive, unless some simplification method were used [2]. Simplification essentially means that, for an update U, it suffices to check only those instances of constraints that are potentially violated by U. That idea is the basis of most methods for integrity checking proposed in the literature or used in practice [2].

Example 1. Let $\mathit{married}(\mathit{Man}, \mathit{Woman})$ be a relation with predicate *married* about married couples in the database of a civil registry (column names are self-explaining). Let I be the denial constraint

$$I \;=\; \leftarrow \mathit{married}(x, y),\, \mathit{married}(x, z),\, y \neq z\,.$$

I states that no man x may be married to two different women y and z, i.e., I forbids bigamy.

Now, let U be an update that inserts $\mathit{married}(\mathit{joe}, \mathit{sue})$. Usually, integrity then is checked by evaluating the following instance I' of I:

$$I' \;=\; \leftarrow \mathit{married}(\mathit{joe}, \mathit{sue}) \wedge \mathit{married}(\mathit{joe}, z) \wedge \mathit{sue} \neq z\,.$$

Since U makes $\mathit{married}(\mathit{joe}, \mathit{sue})$ true, I' can be simplified to

$$I'_s \;=\; \leftarrow \mathit{married}(\mathit{joe}, z) \wedge \mathit{sue} \neq z\,.$$

The simplification I'_s asks if *joe* is married to any person z whose name is not *sue*. Its evaluation essentially amounts to a simple search in the table of *married*, in order to see if there is any woman with name other than *sue* who would be married with *joe*. If so, U is rejected; if not, U can be committed. Clearly, the evaluation of I' is significantly cheaper than the evaluation of I, which would involve a join of the entire *married* relation with itself.

In principle, also the following instance I'' of I would have to be evaluated:

$$I'' \;=\; \leftarrow (\mathit{married}(\mathit{joe}, y) \wedge \mathit{married}(\mathit{joe}, \mathit{sue}) \wedge y \neq \mathit{sue})\,.$$

I'' is obtained by resolving $\mathit{married}(x, z)$ in I with the update $\mathit{married}(\mathit{joe}, \mathit{sue})$. However, no evaluation of I'' is needed since I'' is logically equivalent to I'.

3 Causes of the Violation of Integrity

Informally, causes are minimal explanations of why a constraint is violated. Similarly, causes also may serve as concise justifications of why an answer to a query has been given, and for computing answers that have integrity in inconsistent databases, as shown in [4, 3]. In this paper, we are going to use causes for two purposes: firstly, in Section 4, for quantifying the lack of quality in databases, by sizing sets of causes of integrity violation; secondly, in Section 5, for inconsistency-tolerant integrity checking, by which quality can be controlled.

For simplicity, we contend ourselves in this paper with causes of the violation of definite denials, i.e. constraints without negation in their bodies, and definite databases. Due to the non-monotonicity of database negation, the definition of causes for more general constraints in hierarchical databases is more complicated; it is elaborated in [3].

Definition 2. (*cause*)
Let D be a database and $I = \leftarrow B$ be a constraint.

a) A set E such that each element in E is a ground instance of a clause in D is called an *explanation* of the violation of I in D if there is a substitution θ such that $E \models B\theta$.

b) An explanation C of the violation of I in D is a *cause* of the violation of I in D if no explanation of the violation of I in D is a proper subset of C.

c) E is called a *cause* of the violation of IC in D if there is an $I \in IC$ of which E is a cause.

In part a, $E \models B\theta$ entails that $D \models B\theta$, i.e., I is violated in D. Parts b and c formalize that causes are minimal explanations of the violation of I or, resp., IC in D.

Example 2. Let the constraint $I = \leftarrow \mathit{married}(x, y), \mathit{same\text{-}sex}(x, y)$ be imposed on the database D of Example 1. The predicate *same-sex* be defined in D by $\mathit{same\text{-}sex}(x, y) \leftarrow \mathit{male}(x),\ \mathit{male}(y)$ and $\mathit{same\text{-}sex}(x, y) \leftarrow \mathit{female}(x),\ \mathit{female}(y)$. Clearly, I denies cosexual marriages. Yet, assume $\mathit{married}(\mathit{fred}, \mathit{rory})$ is a fact in D, where both $\mathit{male}(\mathit{fred})$ and $\mathit{male}(\mathit{rory})$ hold, the latter due to a registered gender reversal after marriage. Formally, that amounts to a violation of I. Thus, $\{\mathit{married}(\mathit{fred}, \mathit{rory}), \mathit{male}(\mathit{fred}), \mathit{male}(\mathit{rory})\}$ is a cause of the violation of I in D.

It is easy to see that causes can be obtained as a by-product of standard query evaluation, i.e., the computation of causes is virtually for free.

4 Measuring Quality by Quantifying Causes

Database quality can be quantified by measuring inconsistency, and in particular by quantifying causes of integrity violations. In 4.1, we present a general axiomatization of inconsistency metrics. It enhances the axiomatization in [7]. In 4.2, we introduce a quality metric based on causes. In 4.3, we argue why cause-based inconsistency metrics are preferable to case-based metrics.

4.1 Axiomatizing Inconsistency Metrics

Let $\preccurlyeq$ symbolize an ordering that is antisymmetric, reflexive and transitive. For expressions E, E', let $E \prec E'$ denote that $E \preccurlyeq E'$ and $E \neq E'$.

Definition 3. We say that $(\mu, \preccurlyeq)$ is an *inconsistency metric* (in short, a *metric*) if μ maps pairs (D, IC) to some set that is partially ordered by $\preccurlyeq$, and, for each pair (D, IC) and each pair (D', IC'), the following properties (3) – (5) hold.

$$\text{If } D(IC) = \mathit{true} \text{ and } D'(IC') = \mathit{false} \text{ then } \mu(D, IC) \prec \mu(D', IC') \tag{3}$$

$$\text{If } D(IC) = \mathit{true} \text{ then } \mu(D, IC) \preccurlyeq \mu(D', IC') \tag{4}$$

$$\mu(D, IC) \preccurlyeq \mu(D, IC \cup IC') \tag{5}$$

Property (3), called *violation is bad* in [5, 7], ensures that the measured amount of inconsistency in any pair (D, IC) for which integrity is satisfied is always smaller than what is measured for any pair (D', IC') for which integrity is violated. Property (4), called *satisfaction is best*, ensures that inconsistency is lowest, and hence quality is always highest, in any database that totally satisfies its integrity theory. Property (5) requires that the values of μ grow monotonically with growing integrity theories.

Occasionally, we may identify a metric $(\mu, \preccurlyeq)$ with μ, if $\preccurlyeq$ is understood.

Example 3. A simple example of a coarse, binary inconsistency metric β is provided by the equation $\beta(D, IC) = D(IC)$, with the natural ordering $true \prec false$ of the range of β, i.e., integrity satisfaction $(D(IC) = true)$ means lower inconsistency and hence higher quality than integrity violation $(D(IC) = false)$.

More interesting inconsistency metrics are defined and featured in [5, 7]. In fact, property (5) of Definition 3 has not been discussed in [5, 7], but all examples of inconsistency metrics given there actually satisfy (5) as well.

For instance, the function that maps pairs (D, IC) to the cardinality of the set of cases (instances) of violated constraints is a convenient quality metrics. Inconsistency can also be measured by taking such sets themselves, as elements of the metric set that is constituted by the powerset of all cases of IC, together with the subset ordering. Other metrics can be based on causes of violations, as outlined in the following subsection.

4.2 Cause-based Inconsistency Metrics

The lack of quality in databases can be reflected by counting and comparing sets of causes of the violation of constraints.

Let $\mathsf{CauVio}(D, I)$ denote the set of causes of the violation of I in D, and $\sigma(D, IC) = \{C \mid C \in \mathsf{CauVio}(D,I),\ I \in IC\}$ be the set of all causes of the violation of any constraint in IC. Then, $(\sigma, \subseteq)$ is an inconsistency metric, and so is $(\zeta, \leq)$, where the mapping ζ is defined by $\zeta(D, IC) = |\sigma(D,\ IC)|$ and $|\,.\,|$ denotes set cardinality. In words, ζ counts causes of integrity violation.

Example 4. Let IC consist of the constraint I (no bigamy) in Example 1, and a person named *sheik* be registered as a male citizen. i.e., $male(sheik) \in D$. Further, suppose that also the n facts $married(sheik, wife_1), \ldots, married(sheik, wife_n)$ $(n \geq 2)$ are in D, and that there is no other man in D who is married more than once. Then, for each i, j such that $1 \leq i, j \leq n$ and $i \neq j$, $\{married(sheik, wife_i), married(sheik, wife_j)\}$ is a cause of the violation of IC in D. Hence, the inconsistency in D as measured by ζ is $\zeta(D, IC) = 1 + 2 + \ldots + n{-}1$. Thus, for $n > 3$, the inconsistency as measured by ζ that is caused by a man who is married to n different women is higher than the inconsistency of n men being married to just 2 women.

4.3 Causes vs Cases

As seen in Example 3, integrity and hence the quality of databases can be coarsely measured by checking the integrity constraints imposed on them. In [5, 7], we have shown that this assessment of quality can be further refined by focusing on cases, i.e., instances of constraints that are actually violated. That focus takes into account that the majority of instances of constraints (which typically are universally quantified formulas, in general) remains satisfied, while only certain cases lack integrity and hence suffer from quality impairment.

Still, the association of the quality of a database with constraint violations, or even with violated cases of constraints, does not directly tell which are the actually stored data that are responsible for violating constraints, i.e. for the lack of quality. Hence, measuring quality by quantifying causes is preferable to the case-based approach, as illustrated by the following example.

Example 5. Suppose the predicate p in the constraint $I = \leftarrow p(x, x)$ (which requires the relation corresponding to p to be anti-reflexive) is defined by the two clauses $p(x, y) \leftarrow q(x, y), q(y, x)$ and $p(x, y) \leftarrow r(x, z), s(y, z)$. Further, suppose that the case $I' = \leftarrow p(c, c)$ of I is violated. With that information alone, as provided by focusing on violated constraints, it is not clear whether the violation of I is due to the existence of the tuple $q(c, c)$ in the database or to the existence of one or several pairs of tuples of the form $r(c, z)$ and $s(c, z)$ in the join of r and s on their respective last column. In fact, an arbitrary number of causes for the violation of I and even of I' may exists, but the case-based approach of quantifying the quality of databases does not give any account of that. As opposed to that, the cause-based approach presented in 4.2 clearly does.

Another advantage of causes over cases is that the latter do not provide any means for computing reliable answers to queries in inconsistent databases, while the former do, as shown in [4, 3].

5 Controlling Impaired Quality

In 5.1, we are going to recapitulate from [5, 7] that metric-based inconsistency-tolerant integrity checking can be used to monitor and control the evolution of impaired quality across updates. In 5.2, we are going to define inconsistency-tolerant cause-based integrity checking methods, by which the monitoring and control of quality impairment can be implemented.

5.1 Metric-based Inconsistency-tolerant Integrity Checking

For monitoring and controlling quality impairment in databases, it is desirable to have a mechanism that is able to preserve or improve the quality across updates, while tolerating extant impairments of quality. By Definition 4, below, which generalizes Definition 1, we are going to see that each inconsistency metric induces a sound integrity checking method that provides the desired properies.

Definition 4. (*metric-based inconsistency-tolerant integrity checking*)
Let $\mathcal{M}$ be a method for integrity checking and $(\mu, \preccurlyeq)$ be an inconsistency metric. $\mathcal{M}$ is called *sound*, resp., *complete wrt. metric-based inconsistency tolerance* if, for each triple (D, IC, U), (6) or, resp., (7) holds.

$$\text{If } \mathcal{M}(D, IC, U) = sat \text{ then } \mu(D^U, IC) \preccurlyeq \mu(D, IC). \tag{6}$$

$$\text{If } \mu(D^U, IC) \preccurlyeq \mu(D, IC) \text{ then } \mathcal{M}(D, IC, U) = sat. \tag{7}$$

If (6) holds, then $\mathcal{M}$ is also called *metric-based*, and, in particular, μ*-based*.

Definitions 1 and 4 are structually quite similar. However, there are two essential diferences. Firstly, the premise $D(IC) = true$ in Definition 1 is missing in Definition 4. This premise requires that integrity be totally satisfied before the update U. By contrast, inconsistency-tolerant integrity checking, as characterized by Definition 4, does not expect the total satisfaction of all integrity constraints. Rather, it ignores any extant violations (since the total integrity premise is absent), but prevents that the quality degrades across updates by additional violations, as guaranteed by the consequence of condition (6). So, the second difference to be mentioned is that the consequence of condition (6) clearly weakens the consequence of condition (1), and, symmetrically, the premise of condition (7) weakens the premise of condition (2). Obviously, (1) (resp., (7)) coincides with (1) (resp., (4)) for $\mu = \beta$ (cf. Example 3). If, additionally, $\mathcal{M}$ is a traditional integrity checking method that insists on the total integity premise, then both definitions coincide.

5.2 Cause-based Integrity Checking

Obviously, Definition 4 does not indicate how $\mathcal{M}$ would compute its output. However, for each metric $(\mu, \preccurlyeq)$, condition 8, below, defines a μ-based method, as already proved in [5, 7].

$$\mathcal{M}^{\mu}(D, IC, U) = sat \text{ iff } \mu(D^U, IC) \preccurlyeq \mu(D, IC). \tag{8}$$

Hence, for $\mu = \sigma$ or $\mu = \zeta$, we obtain two sound and complete cause-based inconsistency-tolerant integrity checking methods for controlling quality. That is illustrated by the following example. It also illustrates that different modes or degrees of inconsistency tolerance can be obtained by suitable choices of metrics.

Example 6. Let D and IC be as in Example 4. Further, suppose that *sheik* divorces from $wife_1$, and is about to wed with $wife_{n+1}$, as expressed by the update request $U = \{delete\ married(sheik, wife_1),\ insert\ married(sheik, wife_{n+1})\}$. Thus, $married(sheik, wife_{n+1}) \in D^U$ and $married(sheik, wife_{n+1}) \notin D$, hence $\sigma(D^U, IC) \nsubseteq \sigma(D, IC)$, hence $\mathcal{M}^{\sigma}(D, IC, U) = vio$. On the other hand, we clearly have $\zeta(D^U, IC) = \zeta(D, IC)$, hence $\mathcal{M}^{\zeta}(D, IC, U) = sat$.

6 Conclusion

We have revisited the idea to model the quality of stored data by integrity constraints. We have outlined how to measure the lack of data quality by focusing on causes, i.e., sets of data that are responsible for constraint violations.

In a previous approach that was also based on quality modeled by constraints [5, 7], violated instances of constraints called cases had been measured. The basic idea of measuring quality by causes is very similar to the basic idea of measuring quality by cases: the less/more cases or causes of violated constraints exist, the better/worse is the quality of data. However, we have seen that cause-based metrics quantify the data that lack quality in a more directly and hence more accurately than the metrics based on cases. Moreover, we have shown that quality metrics, and in particular those based on causes, also serve to monitor and control the increase of unavoidable quality impairment.

Apart from our own previous work and the trivial inconsistency measure β, as characterized in Example 3, the ideas of quantifying the quality of databases by measuring the causes of integrity violation and of controlling the evolution of quality by inconsistency-tolerant measure-based integrity checking is original of this paper. Thus, the there is no further related work, except the basic paper on inconsistency measures in databases [8] and the literature on inconsistency tolerance in general, as discussed in [4, 6].

The work presented in this paper essentially is of academical nature. A lot of details remain open for making our theoretical ideas useful in practice. One example of many is given by the question of how to incorporate common practical constructs such as NULL values and aggregate functions into the concept of cause-based inconsistency-tolerant integrity checking. These and other issues are on the agenda for future investigations.

Other upcoming work of ours is going to deal with assigning application-specific weights to causes that violate cases of constraints. The purpose of that is to obtaining quality metrics that reflect the given application semantics in a more dedicated manner. Moreover, we are working on efficient ways to compute causes for the general case of databases and constraints that allow for negation in the body of clauses, as a basis for implementing cause-based inconsistency metrics. Preparatory theoretical studies in that direction have been initiated in [3].

References

1. S. Ceri, G. Gottlob, L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *TKDE* 1(1):146-166, 1989.
2. H. Christiansen, D. Martinenghi. On simplification of database integrity constraints. *Fundam. Inform.* 71(4):371–417, 2006.
3. H. Decker. Answers that Have Integrity in Databases that Violate Constraints. Presented at the ICALP workshop SDKB 2010, to appear in the post-workshop proceedings of SDKB, 2011.

4. H. Decker. Toward a Uniform Cause-based Approach to Inconsistency-tolerant Database Semantics. *Proc. 9th ODBASE, Part II*, pp. 983–998. Springer LNCS vol. 6427, 2010.
5. H. Decker. Quantifying the Quality of Stored Data by Measuring their Integrity. *Proc. DIWT 2009, Workshop SMM*, pp. 823–828. IEEE, 2009.
6. H. Decker, D. Martinenghi: Inconsistency-tolerant Integrity Checking. *TKDE* 23(2):218-234, 2011.
7. H. Decker, D. Martinenghi. Modeling, Measuring and Monitoring the Quality of Information. *Proc. 28th ER, Workshops*, pp. 212–221. Springer LNCS vol. 5833, 2009.
8. J. Grant, A. Hunter. Measuring inconsistency in knowledgebases. *J. Intelligent Information Systems* 27(2):159–184, 2006.
9. R. Ramakrishnan, J. Gehrke. *Database Management Systems*. McGraw-Hill, 2003.