Query Evaluation in Almost Consistent Databases Using Residues

Marcelo Arenas
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
marenas@ing.puc.cl

Leopoldo Bertossi
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
bertossi@ing.puc.cl

Jan Chomicki
Monmouth University
Department of Computer Science
chomicki@moncol.monmouth.edu

Abstract

In this paper we consider the problem of the logical characterization of the notion of correct answer in a relational database that may violate given integrity constraints. This notion is captured in terms of the possible repaired versions of the database. A computational counterpart of the semantical notion is provided in terms of the reconstruction of the database as a deductive database to which concepts and techniques from semantic query optimization are applied with the purpose of computing correct answers.

1. Introduction

In databases it is usually the case that when a transaction is going to violate an integrity constraint, the transaction cannot be executed. Nevertheless, there are cases in which we are willing to accept an inconsistent database because: (1) We know that the DB consistency can be repaired by executing new transactions in the future; (2) We are not in position to detect any violations, maybe because our query language is not expressive enough; (3) The inconsistent database can still give us correct answers to certain queries; ...

In this paper we will consider a form of relaxed consistency of databases, according to which violations are allowed to exist and persist. In this context, it is important to characterize in logical terms the fact that we can trust, hopefully big, parts of the database, still obtaining correct answers to queries. Intuitively, an answer to a query posed to a database that violates some given integrity constraints will be correct in a precise sense: It should be the same as the answer obtained from any repaired version of the original database.

Example 1. Consider a database subject to the *IC*: "the number of items in stock must be at least 30". Assume that according to the database there only 28 items in stock, being the *IC* violated. We are willing to keep working with the database, because: (1) Soon a new supply will bring the database back to a consistent state; (2) Our database can still answer some queries correctly. For example, if we ask "Are there at least 26 items in stock?", the answer will be YES, the same as from any consistent instance of the database.

Example 2. If we have an heterogeneous database system, it is possible that only a certain kind of queries can be asked to a particular database in the system due to the lack of expressive power of the query language. In particular, it is possible that we cannot verify its consistency from outside the database. We will need to trust that database and keep asking queries to it, some of them will be correct wrt to the implicit assumption about its consistency.

In order to make the conceptualization of correct answer precise, it is necessary to deal with the problem of repairing a database, that is with the problem of taking the database to a new, consistent state. For the purpose of defining correct answers in inconsistent databases, we use concepts and techniques borrowed from semantic query optimization [1], that is we use semantic knowledge about the database that is represented by means of integrity constraints. [3] presents an interesting survey of applications of integrity constraints, in particular, semantic query optimization is discussed.

We will also see that our approach to answer correctness allows us to determine alternatives to be considered in order to repair the consistency of the database. It turns out that our subject has also interesting connections to the problem

of detecting violations of integrity constraints by means of queries in a particular language.

We will concentrate mostly on relational databases. Nevertheless, we think that our approach can be extended to the case of deductive databases as well.

2. Some Useful Notions

In this paper we will assume that we have a fixed database schema and a fixed database domain D. We also have a first order language based on this schema with names for the elements of D. Concrete, physical databases will be instances of the schema. We will see them as structures for interpreting the first order language, as such they all share the given domain D, nevertheless, since tables are finite, every instance will have a finite active domain which is a subset of D. We will consider instances as structures compatible with the formal language. There is also a set of integrity constraints, IC, expressed in that language, which the database instances are expected to satisfy. We will assume that IC is consistent in the sense that there is a database instance that makes it true. In IC built–in predicates may appear.

A database instance, r, is consistent if r satisfies IC, that is, $r \models IC$. If r satisfies a subset of IC, we say that r is partially consistent.

2.1. Repairing partially consistent DBs

Given a database instance r, we denote with $\Sigma(r)$ the set of formulas $\{P(\bar{a}) | r \models P(\bar{a})\}$. We can also define a notion of distance between databases instances r and r': $\Delta(r,r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r))$, the symmetric difference. With this we can say that r is closer to r' than to r'' if $\Delta(r,r') \subseteq \Delta(r,r'')$, i.e., if the distance between r and r' is less than the distance between r and r''. We formalize this notion by means of the relation $\leq_r$.

Definition 1. Given a database's instance r, we say that $r' \leq_r r''$ if $\Delta(r,r') \subseteq \Delta(r,r'')$.

Now we will define the notion of a repair of a database instance.

Definition 2. Given database instances r and r', we say that r' is a repair of r if $r' \models IC$ and r' is $\leq_r$*-minimal.*

Example 3. Let us consider a database schema with two tables P and Q, both with one argument. Let the domain contain a, b, c. Assume that for an instance r, $\Sigma(r) = \{P(a), P(b), Q(a), Q(c)\}$, and let $IC = \{\forall x(P(x) \supset Q(x))\}$. As we can see, r does not satisfies IC because $r \models P(b) \wedge \neg Q(b)$.

In this case we have two possibles repairs for r. First, we can falsify $P(b)$ for satisfying IC, obtaining an instance r' with $\Sigma(r') = \{P(a), Q(a), Q(c)\}$. As a second alternative, in order to satisfy IC is possible to make $Q(b)$ true, obtaining an instance r'' with $\Sigma(r'') = \{P(a), P(b), Q(a), Q(b), Q(c)\}$.

The definition of a repair satisfies certain desirable and expected properties. Firstly, a consistent database does not need to be repaired, because if r satisfies IC, then, by the minimality condition wrt the relation $\leq_r$, r is the only repair of itself (since $\Delta(r,r)$ is empty). Secondly, a partially consistent database r can always be repaired because there is a database r' that satisfies IC, and $\Delta(r,r')$ is finite.

2.2. Querying partially consistent DBs

We need to formalize the intuition that even when a database is inconsistent, it is possible to obtain correct answers to queries. This is important if we still want or need to keep working with an only partially consistent database.

Definition 3. We say that a (ground) tuple $\bar{t}$ is a correct answer to a query $Q(\bar{x})$ (a formula of the language) in the database instance r if for every repair r' of r, it holds $r' \models Q(\bar{t})$. If Q is a sentence, then *true* (*false*) is a correct answer to Q in r, if for every repair r' of r, $r' \models Q$ ($r' \not\models Q$).

Example 4. (example 1 continued) The answer YES to the query "Are there at least 26 items in stock?" in the given database that contained only 28 items is correct because in any repair of the database there will be at least 30 items in stock. Of course, in those databases there will be at least 26 items in stock.

On another side, if we pose the query "Are there at least 29 items in stock?", then we obtain the answer NO, that, being true wrt to the instance of database, is not correct wrt to the assumption of consistency of the database. The answer to that query will change its truth values as soon as we repair the database. Notice also that knowing that the answer NO is incorrect would allow us to detect a violation of the ICs.[1]

We would like to have a computational mechanism for determining if an answer to a query is correct or not. Usually a query produces a set of tuples as answers. With such mechanism we would be in position to detect which of the tuples are correct and which are not. In the rest of the paper we address the problem of constructing such a mechanism. We will achieve this goal when we restrict the ICs to be of a certain, but interesting, syntactical form. The computational mechanism will be provably sound.

We will apply ideas and techniques from semantical query optimization [1] as it appears in deductive databases to the construction of the computational counterpart of the semantical notion of correct answer. For doing this we

[1] In the extended version of this paper we present a connection of the problem of qualifying answers with the problem of detecting violations to the ICs by means of queries.

will need to adapt the methodology to the case of relational databases.

3. Applying Semantic Query Optimization in Relational DBs

3.1. A review of SQO

In a deductive database we have an extensional database, *EDB*, and an intensional database, *IDB*. The extensional predicates are the ordinary tables and the intensional predicates are defined by means of logical rules. Another component of the database is a set of integrity constraints, *IC*, which are represented by rules of the form $\leftarrow B_1$, where the body B is a conjunction of literals . These are conditions that are expected to be satisfied by the database along its evolution. They impose a condition on the possible models of the database and in that sense they provide semantic information about the database. The idea of semantic query optimization is to take advantage of these constraints in the process of answering queries, in such a way that the answering process is optimized by reduction of the search space.

According to [1], such a query evaluation procedure can be divided into a semantic compilation step followed by a query transformation step. Semantic compilation can be informally described as the process of retaining relevant fragments of the ICs. After this step, all the information that can be useful for answering queries has been extracted from the ICs. This compilation is done on the relatively stable components of the database, as *EDB*, *IDB* and *IC*, so that it is executed once on the deductive database, independently of any particular query. During this step, the fragments of the ICs, called "residues", are computed and associated to new deductive rules that can be used later on.

When a query is posed, a semantic transformation procedure uses the already stored residues for generating possibly several semantically equivalent queries. Informally, a query is semantically equivalent to the original query if both produce the same answer on the same database. In this way, maybe one or more new queries will be in position to be evaluated faster than the original query.

In the context of the semantic compilation step, the notion of "subsumption" appears. Subsumption means to include or consider something as a particular case of a more general rule or principle. Given rules in *IDB*, that is those rules that define the intentional relations, of the form $R \leftarrow B_2$, in SQO one attempts to subsume the negated bodies of the intentional rules, that is the $\leftarrow B_2$, into the ICs. It is uncommon that such a perfect subsumption is achieved, because this embodies a contradiction, or better, that no tuples appear in the intentional relation. Instead, it is more common that only a subclause of an IC subsumes the negated body. In this case, the discordant fraction, the residue, represents additional information that can be associated to the relation.

Example 5. Consider the IC

$$\leftarrow Disease(type, code), type = A,$$

saying that "There are no diseases of type A" and the intentional relation

$$DiseaseA(type, code) \leftarrow Disease(type, code), type = A,$$

then it is obvious that no tuples can be introduced in the relation, because there is a contradiction between the definition of the table and the IC. In this case, the negation of the body of the defining rule is subsumed by the IC.

Now, if we define a new relation

$$DiseaseT(type, code) \leftarrow Disease(type, code), code > 1000,$$

then, although the IC does not subsume

$$\leftarrow Disease(type, code), code > 1000,$$

a part of it does, namely $\leftarrow Disease(type, code)$.

In this case, the discordant fraction is $\{\leftarrow type = A\}$. This residue can be associated to the relation *DiseaseT* in order to emphasize that fact that diseases of type A are not allowed.

3.2. Generating residues in relational DBs

In this section we show how to accommodate the techniques introduced in SQO for deductive databases, like generation of residues and new rules, to the relational database context. The rules we will generate are not exactly rules as in deductive databases, rather they are computational mechanisms without the clear declarative contents they have in deductive databases. The reason for this difference is the fact that we are considering relational databases only.

The expansion phase. We need the ICs in a canonical form that is suitable for our processing of them.

Definition 4. An integrity constraint is in standard format if it has the form

$$\forall(\bigvee_{i=1}^{m} P_i(\overline{x_i}) \vee \bigvee_{i=1}^{n} \neg Q_i(\overline{y_i}) \vee \psi),$$

where $\forall$ represents the universal closure of the formula, $\overline{x_i}$, $\overline{y_i}$ are tuples of variables and ψ is a formula that mentions

only built–in predicates, in particular, equality. Notice that there are no constants in the P_i, Q_i, if they are needed they can be pushed into ψ.

In the so called "expansion step" the ICs are transformed into logically equivalent formulas in the standard format. In consequence, our syntactical restriction on a set of integrity constraints for the rest of this paper is that it has a logically equivalent set of integrity constraints that are in standard format.

Example 6. Assume that we have the following integrity constraint on the tables P, Q and the element a: $\forall x(P(x,a) \equiv Q(x,a))$. In order to bring the integrity constraint into the standard format, we first split it into the following set of sentences:

$$\{\forall x(P(x,a) \supset Q(x,a)), \forall x(Q(x,a) \supset P(x,a))\}.$$

Secondly, we have to change the element a, in the predicate arguments, by a variable:

$$\{\forall(x,y)(P(x,y) \wedge y = a \supset Q(x,y)), \\ \forall(x,y)(Q(x,y) \wedge y = a \supset P(x,y))\}.$$

After that we eliminate the connectives $\wedge$ and $\supset$:

$$\{\forall(x,y)(\neg P(x,y) \vee Q(x,y) \vee y \neq a), \\ \forall(x,y)(\neg Q(x,y) \vee P(x,y) \vee y \neq a)\}.$$

The residues computation phase. After the expansion of *IC*, rules associated to the database schema are generated. This could be seen as considering an instance of the database as an extensional database expanded with new rules, and so obtaining an associated deductive database where semantical query optimization can be used.

For each predicate, its negative and positive occurrences in the ICs (in standard format) will be treated separately with the purpose of generating corresponding residues and rules.

For each IC in standard format

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x_i}) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y_i}) \vee \psi), \qquad (1)$$

and each positive occurrence of a predicate $P_j(\bar{x_j})$ in it, a residue is generated

$$\bar{Q}(\bigvee_{i=1}^{j-1} P_i(\bar{x_i}) \vee \bigvee_{i=j+1}^{m} P_i(\bar{x_i}) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y_i}) \vee \psi), \qquad (2)$$

where $\bar{Q}$ represents a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{x_j}$.

Once all residues for the positive cases are computed, let us denote them by $R_1(\bar{x_j}), \ldots, R_r(\bar{x_j})$, we create the rule[2]

$$\neg P_j(\bar{x_j}) \longmapsto \neg P_j(\bar{x_j})\{R_1(\bar{x_j}), \ldots, R_r(\bar{x_j})\}.$$

The rule has a procedural contents: in order to compute $\neg P_j$, compute the RHS.

For each negative occurrence of a predicate $Q_j(\bar{y_j})$ in (1), the following residue is generated

$$\bar{Q}(\bigvee_{i=1}^{m} P_i(\bar{x_i}) \vee \bigvee_{i=1}^{j-1} \neg Q_i(\bar{y_i}) \vee \bigvee_{i=j+1}^{n} \neg Q_i(\bar{y_i}) \vee \psi),$$

where $\bar{Q}$ is a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{y_j}$.

Once all residues for the negative cases are computed, let us denote them by $R'_1(\bar{y_j}), \ldots, R'_s(\bar{y_j})$, we create the rule[3]

$$Q_j(\bar{y_j}) \longmapsto Q_j(\bar{y_j})\{R'_1(\bar{y_j}), \ldots, R'_s(\bar{y_j})\}.$$

Notice that there is exactly one new rule for each positive predicate, and exactly one rule for each negative predicate.

Example 7. If we have the following ICs in standard format

$$IC = \{\forall x(R(x) \vee \neg P(x) \vee \neg Q(x)), \forall x(P(x) \vee \neg Q(x))\},$$

the following rules are generated:

$$\begin{array}{rcl} P(x) & \longmapsto & P(x)\{R(x) \vee \neg Q(x)\} \\ Q(x) & \longmapsto & Q(x)\{R(x) \vee \neg P(x), P(x)\} \\ R(x) & \longmapsto & R(x) \\ \neg P(x) & \longmapsto & \neg P(x)\{\neg Q(x)\} \\ \neg Q(x) & \longmapsto & \neg Q(x) \\ \neg R(x) & \longmapsto & \neg R(x)\{\neg P(x) \vee \neg Q(x)\} \end{array}$$

In section 4 we will show how to use these rules for computing correct answers to queries.

The reduction phase. Once the rules have been generated, it is possible to simplify the residues in them. In every new rule of the form $P(\bar{u}) \longmapsto P(\bar{u})\{R_1(\bar{u}), \ldots, R_r(\bar{u})\}$ the auxiliary quantifications introduced in the expansion step are eliminated (both the quantifier and the associated variable in the formula) from the residues by the process inverse to the one applied in the expansion. The same is done with rules of the form $\neg P \longmapsto \neg P\{\cdots\}$.

[2] In deductive databases, we could generate a rule of the form $(\neg P_j)^I(\bar{x_j}) \leftarrow \neg P_j(\bar{x_j})\{R_1(\bar{x_j}), \ldots, R_r(\bar{x_j})\}$, where, to avoid circularity, $(\neg P_j)^I$ is a new intentional predicate associated to the extensional predicate P_j. In the residues R_i only some of the new intentional predicates or built–in predicates should appear.

[3] Again, in a deductive database context we could generate the rule $(Q_j)^I(\bar{y_j}) \leftarrow Q_j(\bar{y_j})\{R'_1(\bar{y_j}), \ldots, R'_s(\bar{y_j})\}$, with only intentional versions of the extensional predicates or built–in predicates in the residues only.

Example 8. (motivated by [7]) In a company database the table $Supply(x,y,z)$ stands for "Company x supplies to department y the item z". The IC

$$\forall(x,y,z)(Supply(x,y,I_1) \supset Supply(x,y,I_2))$$

says that "If a company supplies to a department item I_1, then necessarily it also supplies item I_2". Its version in the standard format is

$$\forall(x,y,z,w)(\neg Supply(x,y,z) \vee Supply(x,y,w) \vee z \neq I_1 \vee w \neq I_2).$$

The following is one of the two rules to be generated

$$Supply(x,y,z) \longmapsto Supply(x,y,z) \{\forall w(Supply(x,y,w) \vee z \neq I_1 \vee w \neq I_2)\}.$$

Simplifying the residue by elimination of the quantification on w, we obtain

$$Supply(x,y,z) \longmapsto Supply(x,y,z) \{Supply(x,y,I_2) \vee z \neq I_1\}.$$

4. An Operator on Queries

In order to determine correct answers to queries in partially consistent databases, we will make use of an operator, the T operator, that will be applied iteratively on a given query. The idea is to apply all the generated transformation rules obtained at the end of the residue computation step.

Definition 5. The application of operator T_n to a query is defined by means of the following rules

1. $\mathrm{T}_n[\Box] := \Box$, $\mathrm{T}_n[\neg\Box] := \neg\Box$, for every $n \geq 0$ ($\Box$ is the empty clause).
2. $T_0[\varphi] := \varphi$.
3. For each $P(\bar{u})$, if there is the rule $P(\bar{u}) \longmapsto P(\bar{u})\{R_1(\bar{u}), \ldots, R_r(\bar{u})\}$, then
$$\mathrm{T}_{n+1}[P(\bar{u})] := P(\bar{u}) \wedge \bigwedge_{i=1}^{r} \mathrm{T}_n[R_i(\bar{u})].$$
If $P(\bar{u})$ does not have residues, then $\mathrm{T}_{n+1}[P(\bar{u})] := P(\bar{u})$.
4. For each $\neg Q(\bar{v})$, if there is the rule $\neg Q(\bar{v}) \longmapsto \neg Q(\bar{v})\{R'_1(\bar{v}), \ldots, R'_s(\bar{v})\}$, then
$$\mathrm{T}_{n+1}[\neg Q(\bar{v})] := \neg Q(\bar{v}) \wedge \bigwedge_{i=1}^{s} \mathrm{T}_n[R'_i(\bar{v})].$$
If $\neg Q(\bar{v})$ does not have any residues, then $\mathrm{T}_{n+1}[\neg Q(\bar{u})] := \neg Q(\bar{u})$.
5. If φ is a formula in prenex disjunctive normal form, that is,
$$\varphi = \bar{Q} \bigvee_{i=1}^{s} (\bigwedge_{j=1}^{m_i} P_{i,j}(\bar{u}_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg Q_{i,j}(\bar{v}_{i,j}) \wedge \psi_i),$$
where $\bar{Q}$ is a sequence of quantifiers and ψ_i is a formula that includes only built–in predicates, then for every $n \geq 0$:
$$\mathrm{T}_n[\varphi] := \bar{Q} \bigvee_{i=1}^{s} (\bigwedge_{j=1}^{m_i} \mathrm{T}_n[P_{i,j}(\bar{u}_{i,j})] \wedge \bigwedge_{j=1}^{n_i} \mathrm{T}_n[\neg Q_{i,j}(\bar{v}_{i,j})] \wedge \psi_i).$$

Notice that the application of the T operator to a formula produces a new formula. Additionally, $\mathrm{T}_\omega[\varphi]$ will store, in a possibly infinite set, the results $\mathrm{T}_n[\varphi]$, for every $n \geq 0$.

Definition 6. The application of operator T_ω on a query is defined as $\mathrm{T}_\omega[\varphi] = \bigcup_{n<\omega} \{\mathrm{T}_n[\varphi]\}$.

In the rest of this section, we will consider queries $Q(\bar{x})$ written in prenex disjunctive normal form.

Example 9. (example 7 continued) For the query $Q(x) : \neg R(x)$ we have $\mathrm{T}_1[Q(x)] = \neg R(x) \wedge (\neg P(x) \vee \neg Q(x))$, $\mathrm{T}_2[Q(x)] = \neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))$ and $\mathrm{T}_3[Q(x)] = \mathrm{T}_2[Q(x)]$. We have reached a fix point and then $\mathrm{T}_\omega[Q(x)] = \{\neg R(x) \wedge (\neg P(x) \vee \neg Q(x)), \neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))\}$.

We want to use the T operator for answering queries in almost consistent databases. First we need some results, in particular, that the operator works properly when we try to answer queries in a consistent database.

Proposition 1. Given a database instance r and a set of integrity constraints IC, such that $r \models IC$, then for every query $Q(\bar{x})$ and every natural number n: $r \models \forall\bar{x}(Q(\bar{x}) \equiv \mathrm{T}_n[Q(\bar{x})])$.

Corollary 1. Given a database instance r and a set of integrity constraints IC, such that $r \models IC$, then for every query $Q(\bar{x})$ and every tuple $\bar{t}$: $r \models Q(\bar{t})$ if and only if $r \models \mathrm{T}_\omega[Q(\bar{t})]$.

Now we will show the relationship between the correctness of an answer to a query wrt to a database instance r (definition 3) and the answer given to the application of the T operator to the query.

Proposition 2. Given a database instance r, a set of integrity constraints IC and a query $Q(\bar{x})$, if $r \models \mathrm{T}_\omega[Q(\bar{x})](\bar{t})$, then $\bar{t}$ is a correct answer to Q in r (in the sense of definition 3).

We can see that the operator can be used to be sure that some of the answers obtained by direct querying of the inconsistent database are correct, all we need is to see if they

appear in the answer set to the transformed query obtained by application of the T operator.

Example 10. (motivated by [7]) Consider a database with the following IC, telling that C is the only supplier of items of class T_4: $\forall(x,y,z)(Supply(x,y,z) \wedge Class(z,T_4) \supset x = C)$, which transformed into the standard format is

$$\forall(x,y,z,w)(\neg Supply(x,y,z) \vee \neg Class(z,w) \vee w \neq T_4 \vee x = C).$$

The following rule can be generated:

$$Class(z,w) \longmapsto Class(z,w) \{\forall(x,y)(\neg Supply(x,y,z) \vee w \neq T_4 \vee x = C)\}.$$

Consider now the following database that violates the IC:

Supply		
C	D_1	I_1
D	D_2	I_2

Class	
I_1	T_4
I_2	T_4

If we pose the query $Class(z,T_4)$?, asking for the items of class T_4, directly to the database, we obtain I_1 and I_2. Nevertheless, if we pose the query $T_\omega[Class(z,T_4)]$, that is

$$Class(z,T_4) \wedge \forall(x,y)(\neg Supply(x,y,z) \vee x = C),$$

we obtain only I_1, eliminating I_2, whose supplier is not C. That is, only I_1 is to be considered as a correct answer.

Example 11. (example 8 continued) We had the following IC:

$$\forall(x,y,z)(Supply(x,y,I_1) \supset Supply(x,y,I_2)),$$

saying that item I_2 is supplied whenever item I_1 is supplied. The rules were already generated in example 8. Consider now the following inconsistent instance of the database

Supply		
C	D_1	I_1
C	D_1	I_3

and pose the query $Supply(C,D_1,z)$?, asking for the items supplied by C to D_1. The answer from the database instance is I_1 and I_3. There should be something wrong with item I_1, because item I_2 is not supplied. If we compute now $T_\omega[Supply(C,D_1,z)]$, we obtain a set of queries that is equivalent to the formula

$$Supply(C,D_1,z) \wedge (Supply(C,D_1,I_2) \vee z \neq I_1),$$

thus if we ask $T_\omega[Supply(C,D_1,z)]$, we obtain only I_3 as a correct answer.

Example 12. Consider now a student database. $Student(x,y,z)$ means that x is the student number, y is the student's name, and z is the student's address. The two following ICs state that the first argument is a key of the relation

$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset y = u),$$
$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset z = v).$$

In the standard format, these ICs take the form

$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee \neg Student(x,u,v) \vee y = u),$$
$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee \neg Student(x,u,v) \vee z = v).$$

The following rule can be generated

$$Student(x,y,z) \longmapsto Student(x,y,z) \{\forall(u,v)(\neg Student(x,u,v) \vee y = u), \forall(u,v)(\neg Student(x,u,v) \vee z = v)\}.$$

Given the database instance

Student		
S_1	N_1	D_1
S_1	N_2	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

We pose the query $Course(S_1,y,z)$?, asking for the names of the courses and their grades of the student with number S_1, obtaining (C_1,G_1) and (C_2,G_2). If we pose the query $T_\omega[Course(S_1,y,z)] = Course(S_1,y,z)$, obviously we obtain the same answer. Intuitively, in this case the T operator help us establish that even when the name of the student with number S_1, it is still possible to obtain the list of courses in which he/she is registered.

If we pose the query $\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))$, about the courses and grades for a student with name N_1, to the database, we obtain (C_1,G_1) and (C_2,G_2) again. Nevertheless, if we ask $T_\omega[\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))]$?:

$$\exists(u,v)(Student(u,N_1,v) \wedge \forall(y',z')(\neg Student(u,y',z') \vee y' = N_1) \wedge \forall(y',z')(\neg Student(u,y',z') \vee z' = v) \wedge Course(u,x,y))$$

we obtain the empty set of tuples. This answer is intuitively correct, because the number of the student with name N_1 is uncertain, and in consequence it is not possible to find out in which courses he/she is registered. This is because the *Course* table is indexed by the student number.

5. Conclusions and Further Work

This paper represents a first step in the development of a whole research area we have identified around databases that do not fully satisfy expected integrity constraints. In the extended version of this paper we will present applications of the T operator to the problem of determining which part of a partially consistent database does not have to be repaired if one wants to take the database to a consistent state. One can say that applying the T operator to a query is equivalent to posing the original query to that "consistent" part of the original database.

We are also working on the problems of: (1) Detecting violations of integrity constraints by means of given query languages. (2) Introducing degrees of partial satisfactions of integrity constraints. (3) Developing a modal approach to the notion of correct answer in partially consistent databases (notice the modal look of definition 3). Connections to modal approaches to database queries [4] and deontic logic approaches to integrity constraint satisfaction [5] are being established. Those results and full proofs for the propositions will be given in the extended version of this paper.

It seems interesting to establish connections of our results with some work already done by the database community. Some related issues have to do with: (1) Querying a database by means of restricted query languages [4, 6, 8]. (2) Characterizing consistency repair as done in [2].

Acknowledgments

This research has been partially supported by FONDECYT Grants (1971304 & 1980945) and a NSF Grant (IRI-9632870). Part of this research was done when the second author was on sabbatical at the Technical University of Berlin (CIS Group) with the financial support from DAAD and DIPUC.

References

[1] U. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[2] M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*. PhD thesis, Universität Hannover, 1996.

[3] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity Constraints: Semantics and Applications. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer Academic Publishers, 1998.

[4] T. Imielinski. Relative Knowledge in Distributed Database (Extended Abstract). In *Proc. Symposium on Principles of Database Systems, PODS'87*, pages 197–209, 1987.

[5] K. Kwast. A Deontic Approach to Database Integrity. *Annals of Mathematics and Artificial Intelligence*, 9:205–238, 1993.

[6] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *Proc. Symposiun on Principles of Database Systems (PODS'95)*, pages 95–104. ACM Press, 1995.

[7] J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18:227–253, 1982.

[8] A. Rajaraman, Y. Sagiv, and J. Ullman. Answering Queries Using Templates with Binding Patterns. In *Proc. Symposiun on Principles of Database Systems (PODS'95)*, pages 105–112. ACM Press, 1995.