A Review of Integrity Constraint Maintenance and View Updating Techniques

Enric Mayol, Ernest Teniente

Universitat Politècnica de Catalunya
E-08034 Barcelona - Catalonia
e-mail: [mayol | teniente]@lsi.upc.es

Abstract

Two interrelated problems may arise when updating a database. On one hand, when an update is applied to the database, integrity constraints may become violated. In such case, the integrity constraint maintenance approach tries to obtain additional updates to keep integrity constraints satisfied. On the other hand, when updates of derived or view facts are requested, a view updating mechanism must be applied to translate the update request into correct updates of the underlying base facts.

This survey reviews the research performed on integrity constraint maintenance and view updating. It is proposed a general framework to classify and to compare methods that tackle integrity constraint maintenance and/or view updating. Then, we analyze some of these methods in more detail to identify their actual contribution and the main limitations they may present.

Categories and subject descriptors: H.2.4 [**Database Management**]: Systems---Rule-based Databases
Words and Phrases: Updates, Integrity Constraints, Integrity Constraint Maintenance, View Updating

1. Introduction

Most databases, like relational and deductive ones, allow the definition of intentional information. The most traditional types of intentional information are views and integrity constraints. It is widely accepted to define this intentional information in a declarative way, because it provides a uniform representation that allows tools and techniques based on logic programming to be incorporated into database management systems.

Views are defined by means of derivation rules that allow the definition of new facts (view or derived facts) from other derived or stored (base) facts. The view extension is completely defined by the application of derivation rules to the contents of the database. Views provide several advantages like simplifying user interface, favoring logical data independence, or definition of new queries or new intentional information. However, these advantages can only be achieved if a user does not distinguish between derived and base facts.

Integrity constraints define, by means of rules, those conditions that each database state must satisfy. They determine legal states and legal transitions of the database. Integrity constraints are used to prevent the entering of incorrect data into the database and to ensure that database facts satisfy those conditions. Therefore, an integrity constraint is a closed query that must always be true after a database update. Integrity constraints use to be defined in denial form expressing more directly the non-legal states of the database. Integrity constraints defined in other forms can be easily transformed into denials by using the transformation defined in [LlT84].

Views are used to generate new (derived) data from stored facts of the database, while integrity constraints are used to verify that base and derived data satisfy them without generating new data.

This survey relies on the problems of view updating and of maintaining integrity constraints satisfied. We identify the relevant features to be considered when solving these problems. We also summarize the achievements of several methods according to these features, and we analyze some of these methods in more detail to identify their actual contribution and the main limitations they may present.

Early surveys on integrity constraint maintenance and view updating can be found in [FP93, GA93, MT99b]. This survey extends a preliminary version of the survey we presented in [MT99b] by considering new methods and new features that we did not consider there. We also extend the review presented in [FP93] by considering alternative features and more recent methods.

This survey is structured as follows. Section 2 defines more precisely integrity constraint maintenance and view updating problems and analyzes the relationship between both problems. Section 3 makes a brief review of methods that tackle these problems in the relational database model. In Section 4, we define the general framework we use to classify and compare relevant work in this field. The framework is defined in terms of the relevant dimensions to be taken into account during view updating and integrity constraint maintenance. Sections 5 and 6 rely on integrity constraint maintenance. In Section 5, we analyze in more detail specific features of the framework dimensions relative to this problem. Then, in Section 6, we classify and compare some of the best known methods in this field according to these features. Drawbacks and individual limitations that some methods present are also discussed. Section 7 and 8 follow the same structure than sections 5 and 6, but in this case, they consider view updating and integrity constraint maintenance problems altogether. Finally, Section 9 summarizes the main conclusions and points out interesting aspects for further research.

2. Integrity Constraint Maintenance and View Updating

Databases are updated through the application of *transactions* that consist of a set of updates of base facts. Several problems may arise when updating a database [TU95]. Some of them are directly related to updates that involve views or integrity constraints. Therefore, database management systems must provide mechanisms to deal with updates of view facts and mechanisms to ensure that integrity constraints are satisfied after the application of a transaction.

2.1 Integrity Constraint Maintenance

The problem of ensuring that after the application of a transaction all integrity constraints are satisfied, is usually known as *integrity constraint enforcement*. There are several approaches to ensure that a database satisfies integrity constraints [Win90]. All of them are reasonable and the correct approach to be considered depends on the semantics of the integrity constraints and of the database. The best known approaches are integrity constraint checking and integrity constraint maintenance.

Integrity constraint checking is the classical approach to deal with integrity constraints. It is the most conservative one because if integrity constraint becomes violated by the application of a transaction, then the transaction is rejected and the database remains unchanged. An important drawback of this approach is that the user may not have information about possible changes to be made to the transaction to make it obey the integrity constraints.

An alternative approach, aimed at overcoming this limitation, is that of *integrity constraint maintenance*. This approach is concerned with trying to identify additional updates (i.e. repairs) to add to the original transaction so that the resulting transaction does not violate any integrity constraint. Usually, there are several ways to repair an integrity constraint.

Example 2.1: Consider a database with two base facts stating that Ann is a professor and a doctor. Additionally, there is an integrity constraint (Ic) defining that it is not possible to be a professor without being doctor.

Prof(Ann)   Doctor(Ann)

$Ic \leftarrow Prof(p) \land \neg Doctor(p)$

Consider a transaction to delete Ann as a doctor T={delete(Doctor(Ann))}. This transaction violates integrity constraint Ic because, in the new state database, Ann will remain as a professor but not being a doctor. Therefore, to ensure that integrity constraint remains satisfied, we could consider any integrity constraint enforcement approach. By considering the integrity constraint checking approach, transaction T should be rejected, the database would remain unchanged, but the requested deletion could not be satisfied. In contrast, with integrity constraint maintenance the requested deletion could be satisfied without violating the integrity constraint. In this case, we should extend transaction T with the deletion of Ann as a professor. Therefore, the extended transaction T'={delete(Doctor(Ann)), delete(Prof(Ann))} would allow the initial request to be satisfied in the new state, without violating the integrity constraint.

Research performed in the integrity constraint checking field is mainly oriented to define mechanisms to identify, as efficiently as possible, when an integrity constraint becomes violated by a given transaction. In contrast to this approach, methods that follow the integrity constraint maintenance approach are mainly concerned with the generation of alternative ways to ensure integrity constraints satisfaction, and only a few of them explicitly consider efficiency issues.

In this survey, we do not consider methods of integrity constraint checking. Some surveys as [CGMD94] and [Sel95] summarize the research performed in this field. The first one [CGMD94], makes a comparison between concrete methods of integrity constraint checking, while [Sel95] includes in its proposal an interesting comparison of different approaches to tackle the integrity constraint checking problem.

The difficulty and necessary effort to tackle the integrity constraint enforcement problem depends basically on the number of integrity constraints to enforce, on the definition of integrity constraints and on the selected approach to enforce them. However, an additional difficulty may appear when views (or derived information) are involved in the requested update or in the definition of integrity constraints, since derived information must be appropriately managed.

2.2 View Updating

An *update request* U consists of a set of base and derived updates. Update requests are usually asked by the user to change the database or they are demanded to repair an integrity constraint violation. When an update request U contains updates of view/derived facts, the problem of *view updating* appears. View updates can not be directly applied to the database and they must be translated into appropriate updates of the underlying base facts. The obtained set of base fact updates is known as a *translation* of the view update request. Each translation defines a possible transaction that guarantees that when applied to the database, the requested view update becomes satisfied.

Example 2.2: Consider a database with two view predicates: Works(p,u) and Phd-Std(p,u). A person works in a university if s/he is an employee in some department of that university. Phd students of a university are those that work in the university, but do not have a doctor's degree.

Emp(Cris,IS)   Emp(Peter,IS)   Dept(IS,UPC)   Doctor(Peter)

Works(p,u) $\leftarrow$ Emp(p,d) $\wedge$ Dept(d,u)

Phd-Std(p,u) $\leftarrow$ Works(p,u) $\wedge$ $\neg$ Doctor(p)

Consider the update request to insert that Mercè works at the UPC: U={insert(Works(Mercè,UPC))}. A possible way to satisfy this view update request is by the translation T={insert(Emp(Mercè,IS))}.

There are some aspects related to the process of translating a view update request that makes view updating a difficult problem [Ten00].

I. Multiple Translations. Usually, there are several translations to a view update request. For example, consider the update request U={delete(Works(Cris,UPC))} to apply in the database of example 2.2. There are two translations that satisfy U, these are T1={delete(Emp(Cris,IS))} and T2={delete(Dept(IS,UPC))}. Methods for view updating should be able to obtain all alternative translations to an update request. Furthermore, it would be desirable that these methods define some criteria to select the "best" translation.

II. Side effects. A translation of a view update request could induce additional updates in the same or in other views. These non-requested updates are known as side effects, and they are usually hidden to the user. In the previous example, the application of translation T2 induces side effects S={delete(Works(Peter,UPC)), delete(Phd-Std(Cris,UPC))}. In some cases, these side effects could be inappropriate so, methods for view updating must provide some mechanism to prevent them.

III. Non-monotonic. When a view is defined by means of negated atoms, insertions (deletions) of view facts can be satisfied by means of deletions (insertions) of base facts. In the example 2.2, a request to insert view fact Phd-Std(Peter,UPC) may be satisfied by the deletion of base fact Doctor(Peter).

IV. Multiple view update requests. When an update request U contains more than one requested update, all of them must be satisfied all together. However, translating each one independently and combining the obtained translations may not always satisfy the update request at all. Consider example 2.2 and the update request U={insert(Phd-Std(Tom,UPC)), insert(Doctor(Tom))}. A translation of the first request in U is T3={insert(Emp(Tom,IS)} and the second request one is satisfied by transaction T4={insert(Doctor(Tom))}. Notice that transaction obtained by combining both transactions T3$\cup$T4={insert(Emp(Tom,IS)), insert(Doctor(Tom))} does not satisfy U. The reason is that insertion of Doctor(Tom) dismisses the desired effect of insert(Phd-Std(Tom,UPC)). Notice that in this example, it is not possible to satisfy U by only changing extensional database.

V. Translation of existential views. An existential view is defined by means of a derivation rule that contains variables in the body that do not appear in its head. When an update on such views is requested, there may be as many translations as different valid values can be assigned to the existential variables. For example, given the update request of inserting the view fact Works(Joan,UPC) into database of example 2.2, there is a translation T_1={insert(Emp(Joan,IS))} that satisfy the requested update, and there are also as many translations T_i={insert(Emp(Joan,$dept_i$)), insert(Dept($dept_i$,UPC))} as different departments $dept_i$ could have the UPC university. To obtain these translations, a method must define how to obtain valid values for the existential variables. Note that when the domain of existential variables is infinite, there may exist an infinite number of translations.

View updating methods must take into account the above aspects in order to properly deal with view updates. Methods that do not explicitly consider some of these aspects may limit its effectiveness or may suffer some drawback. Some examples of them are shown in Section 8.

2.3 Integrity Constraint Maintenance and View Updating

View updating and integrity constraint enforcement are strongly related. On one hand, a transaction corresponding to a translation of a view update request may violate some integrity constraint. Therefore, view updating requires integrity constraint enforcement. On the other hand, when integrity constraints are defined in terms of views, view definitions must be considered to enforce integrity constraints, since if a violation is detected, it may be repaired by means of a view update request. Therefore, integrity constraint maintenance requires also view updating.

Example 2.3: Consider a database with two view predicates: *Doctor(p)* defining that a person p is a doctor if s/he has written a PhD-Thesis and has passed the PhD-exam; and *ResCert(p)* stating that a person has a research certificate if s/he has written a PhD-Thesis.

Integrity constraints *Ic1*, *Ic2* and *Ic3* state, respectively, that it is not possible to have a research certificate without been author of some good research paper; that it is not allowed to be professor and not to be doctor; and that it is not possible to pass satisfactorily the PhD-exam if some errors were made during the examination.

PassEx(Bob)	$Ic1 \leftarrow ResCert(p) \wedge \neg\, GoodPap(p)$
$Doctor(p) \leftarrow PhD(p) \wedge PassEx(p)$	$Ic2 \leftarrow Prof(p) \wedge \neg\, Doctor(p)$
$ResCert(p) \leftarrow PhD(p)$	$Ic3 \leftarrow Errors(p) \wedge PassEx(p)$

Consider the update request U={insert(Prof(Bob))}. It violates integrity constraint Ic2, so it should be repaired by requesting the insertion of Doctor(Bob). This view update request is translated to the transaction T={insert(PhD(Bob))}. Moreover, the above insertion violates integrity constraint Ic1 since derived fact ResCert(Bob) becomes satisfied. To repair this violation, the insertion of base fact GoodPap(Bob) is required. In this example, we have shown that integrity constraint maintenance requires view updating, and at the same time, translations of a view update request require to maintain integrity constraints satisfied.

As shown in [TO95], view updating and integrity constraint checking can be successfully performed as two separate steps. In a first step, all translations of the view update request are obtained and, in a second step, those translations that violate some integrity constraint are discarded. In contrast, view updating and integrity constraint maintenance cannot be performed in two separate steps, unless additional information of the view updating process is provided to the integrity constraint maintenance process. The reason is that, a repair of an integrity constraint violation may invalidate an already satisfied view update. If this information is not taken into account during the integrity constraint maintenance, it is not possible to guarantee that the obtained translations really satisfy the update request without violating integrity constraints. In the following example, we illustrate this situation.

<u>Example 2.4</u>: Consider the database of example 2.3 and the update request U={insert(Doctor(Bob)), insert(Errors(Bob))}. As in the previous example, the view update insert(Doctor(Bob)) is achieved by insert(PhD(Bob)). Integrity constraints Ic1 and Ic3 are violated and they must be repaired with insert(GoodPap(Bob)) and delete(PassEx(Bob)), respectively. Therefore, if view updating and integrity constraint maintenance are considered separately, the following transaction T={insert(PhD(Bob)), insert(GoodPap(Bob)), insert(Errors(Bob)), delete(PassEx(Bob))} maintains integrity constraints satisfied.

Notice, however, that after applying transaction T, the initial update request U is not satisfied. The deletion of fact PassEx(Bob) invalidates the insertion of derived fact Doctor(Bob). In this sense, during integrity constraint maintenance, we must ensure that the insertion of the view fact Doctor(Bob) is not dismissed by the repair of an integrity constraint.

3. Antecedents in Relational Databases

Relational databases allow the definition of views and integrity constraints. Although views and integrity constraints that can be handled by most of the relational database management systems are limited, several work has been performed in this field. In this section, we review the most significant approaches to deal with view updating and integrity constraint enforcement in relational databases.

Two different approaches are considered to classify the different proposals that tackle the view updating problem.

A first approach is based on considering views as an *abstract data type*. In this sense, definition of a view incorporates the updates that may be requested on that view and the description how these updates should be translated to updates of underlying relations. Methods that follow this approach are [TFC83, FC85, MW88, SM89].

A second approach is based on the definition of a general procedure (*Translator*) that translates a view update request into updates of database relations. The input of this procedure is the view update request, the view definition and the state of the relational database. The output is the set of updates to apply to relational tuples in order to satisfy the requested view update. Methods that follow this approach can be classified in three groups:

– ***Methods based on the complement of a view***

F. Bancilhon and N. Spyratos [Ban79, Spy80, BS81] propose a method for translating view update requests. Given a view definition V, this method defines the complementary view of V such that the complete database can be computed from the view and its complement. Then, a view update request is translated in such a way that only the view V is changed while its complement remains unchanged.

The main contributions of this approach consist in stating that given a view, its complement always exists. Moreover, given a view and its complement, if there is a translation satisfying a view update without changing the complement, this translation is unique. Given a view update request, [BS81] proposes a mechanism to compute this translation.

Main difficulties of this approach rely on its implementation in the relational database model, since the problem of obtaining the minimal complement to a given view is NP-complete [CP84]. Another limitation of this approach [KU84, Kel87] is that, in some cases, the condition of not changing the complement of a view could be too restrictive and, therefore, some valid translations may not be obtained.

– ***Methods defining a specific translator to each view***

For each view definition operator (selection, projection, join, …) and for each update operator (insert, delete, update), methods that follow this approach define a rule specifying the updates to apply to the base relations in order to satisfy a view update request.

In the Table 1, we summarize several proposals that follow this approach. Translation rules for the modification update operator [FSS79, Kel85, Dat86], for the cartesian product operator [FSS79, CA79] and for the intersection operator [LS91] are not included in Table 1. They may be obtained from the respective references.

View operator \ Update operator	**Insertion**	**Deletion**
Projection: V = R [X] [FSS79, Mas84, Kel85, Dat86, LS91]	Insert into R	Delete from R
Selection: V = R (XΘY) [FSS79, Mas84, Kel85, Dat86, LS91]	Insert into R or Modify select. attrib.	Delete from R or Modify select. attrib.
Union: V = R ∪ S [FSS79, Mas84, Kel86, LS91]	Insert into R or Insert into S or Insert into both	Delete from R and Delete from S
Difference: V = R - S [Mas84, LS91]	Insert into R and Delete from S	Delete from R or Insert into S or both of them
Join: V = R [X=Y] S [Kel85, Dat86, LS91]	Insert into R and Insert into S	Delete from R or Delete from S or both of them

Table 1. Summary of view updating methods in relational databases

Notice that the above rules are defined by considering updates of one tuple and views defined by only one operator. In [CA79], it can be found how to manage more complex views and multiple update requests by combining the above rules.

– ***Methods based on the universal relation assumption***

The universal relation assumption [Sag83] consists in viewing a relational database as only one (universal) relation defined by all attributes of the relational database.

Y. Sagiv [Sag83] proposes to consider a *representative instance* of a database as a correct representation of the stored data. This representative instance is automatically obtained from database relations and their functional dependencies.

In this approach, methods to update the database [Sag83, BV88, Lan90] specify update requests in terms of attributes of the representative instance. Then, update requests correspond to projections of the representative instance.

Methods that follow this approach define algorithms to translate insertion, deletions and modifications of the representative instance into updates of the underlying actual database relations.

Relational database management systems actually support some view updates, but in a very limited way, since they can only handle specific types of view definitions. Limitations on view definition are the main reasons because most recent research on the view updating field is performed by considering more expressive definition languages and database models like first order logic and deductive databases.

An interesting overview of the research performed regarding the enforcement of integrity constraint in relational database systems is proposed in [GA93]. This survey begins with the identification of the most relevant research issues of the integrity constraint enforcement problem. Later, different RDBMS prototypes and products that provide some support to handle integrity constraints are described and analyzed.

The authors of this survey conclude that there is still a large gap between the theoretic approach to constraint enforcement and the actual implementations of RDBMS. This is because most implementations are limited to handle only some types of integrity constraints, do not support full transaction semantics and because only little attention has been devoted to the performance of the integrity constraint enforcement process. Although RDBMSs have improved their integrity constraint enforcement functionality, there is still a gap between the research on relational databases in this field and its implementation in RDBMS.

4. Relevant Dimensions of View Updating and Integrity Maintenance

In this section, we define the dimensions we have considered to classify and compare most relevant research performed on view updating and integrity constraint maintenance. In the following sections, we distinguish methods that tackle the integrity constraint maintenance problem from those methods that tackle also the view updating problem. For both groups of methods, we identify the relevant features of each dimension and we classify and compare methods with respect to them.

Given an update request, methods proposed for view updating and integrity constraint maintenance are attempted to obtain solutions to the concrete problem they address by applying the procedure they define. With respect to this procedure, we distinguish the basic mechanism that defines the method from the techniques that it may incorporate to improve its efficiency. With respect to the obtained solutions, we analyze if all solutions may be obtained and if they are sound.

Therefore, the main dimensions to take into account during the process of view updating and integrity constraint maintenance are the following: the problem, the database, the update request, the procedure (basic mechanism, efficiency) and the solutions (soundness, completeness). These dimensions are shown in the following figure:

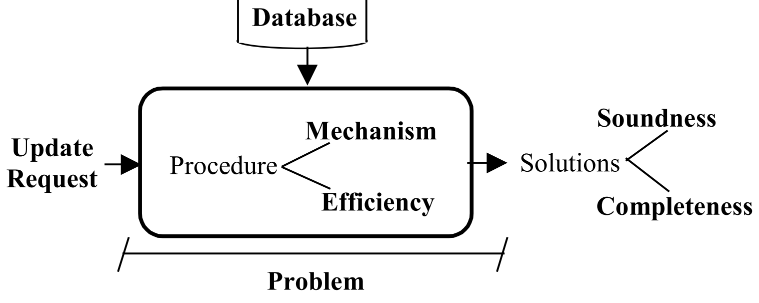

Figure.1 Relevant dimensions of our comparison framework

Problem: In this dimension, we take into account the problem addressed by each method and the approach it follows. The considered problems are integrity constraint maintenance and view updating. We differentiate between methods that tackle integrity constraint maintenance without considering view updating from those methods that consider both problems in an integrated way.

Database: The difficulty of maintaining integrity constraints and managing view updates depends on the definition of the intentional information and on the database model. In this dimension, we include the most relevant features related to how views and integrity constraint are defined in the database schema. We put special attention to the definition language and to the limitations imposed on the views and integrity constraints handled by each method.

Update Request: An important aspect that determines the usefulness of a method is the flexibility a user has to specify its update requests. That is, how many update operators s/he may use to specify an update request or whether update requests are restricted to some specific request types. In this dimension, we analyze some features related to the update request specifications.

Basic Mechanism: Most part of proposals in this research area consist in the formal definition of a mechanism that a method uses to maintain integrity constraints and to translate view update requests. Usually, implementation details are not considered in these definitions. In this dimension, we include the main features related to the basic mechanism without considering implementation issues.

Efficiency: Some methods take explicitly into account efficiency issues in addition to the above basic mechanism description. In this sense, they propose an implementation architecture, or at least, some optimization techniques to improve the efficiency of the basic mechanism. In this dimension, we analyze which efficiency issues are explicitly considered by each method, as well as the efficient techniques these methods propose.

Soundness: It should be expected that solutions obtained by each method satisfy the problem addressed by that method. However, this is not always the case. It may happen that a method is not sound since it may obtain a "solution" that it is not really a solution because it does not satisfy the requested update or the integrity constraints. In this dimension, we analyze whether a method is proved to be sound or, if there are examples showing its unsoundness.

Completeness: In general, several solutions that satisfy an update request may exist. In this dimension, we consider whether a method is able, or not, to obtain all possible solutions.

5. Relevant Features for Integrity Constraint Maintenance

In this section, we identify the main features of each dimension previously defined in Section 4 that are specific to the problem of maintaining integrity constraints. In this sense, we detail the above framework to compare and classify methods that tackle integrity constraint maintenance and that do not explicitly deal with view update requests. These features will be considered again in section 7 when comparing methods for view updating and integrity constraint maintenance.

The relevant features of each dimension that we have taken into account are:

Problem:

Since all the methods considered in this section tackle the same problem, we take into account only one feature of the problem.

- *Run/Compile-Time Approach*: Methods differ in the way they identify violations and propose repairs of integrity constraints. *Run-time* approach knows the actual requested update and the contents of the database. Therefore, it determines the actual repair when actual violations are detected. In contrast, *Compile-time* approach only knows the database schema and a parameterized update request. In this sense, Compile-time approach is intended to generate a program that executed at run-time, when actual values are given to the formal parameters, will provide the desired solutions. However, several methods combine both approaches by performing some preparatory work at compile time before determining actual repairs.

Database Schema:

Methods for integrity constraint maintenance differ considerably with respect to the integrity constraints they may deal with. In this dimension, we have considered four features relative to the expressive power of the database and, more concretely, of the integrity constraints definition.

- *Definition Language*: Several languages can be used to define the database schema and, in particular, to specify integrity constraints. It is important to know the nature of the language used to define a database schema in order to determine the expressive power of the database and, in particular, of integrity constraints. In tables 2 and 3, we indicate for each method the database definition language they use.
- *View definition*: This feature refers to whether view or derived predicates can be defined in the database schema. Some methods define views to improve expressive power of integrity constraints, although they do not directly handle repairs through view updates. In tables 2 and 3, we indicate with *Yes* the methods that allow the use of read-only views.
- *Integrity Constraint Definition*: With this feature, we analyze in more detail the syntactic restrictions that each method imposes on the definition of the integrity constraints. These particular restrictions limit the integrity constraints that a method can handle. In tables 2 and 3, we indicate with *Limited* those methods that impose some restriction on integrity constraints; with *Yes* those that do not impose any restriction and with *Flat* those that do not allow view predicates to appear in integrity constraint definitions.

– Integrity Constraint Types: This feature states whether a method may handle static integrity constraints only or if it can handle also dynamic integrity constraints. Static integrity constraints impose conditions involving only a certain state of the database. Dynamic integrity constraints impose conditions involving more than one state. In tables 2 and 3, we distinguish methods by means of the *Static* and *Dynamic* label.

Update Request:

All methods considered in this section allow the definition of multiple update requests, that is, update requests that involve more than one update. A distinguished feature as far as the update request is concerned is the following:

– Update Operators: Three different kinds of update operators are usually considered: insertions, deletions and modifications. We distinguish those methods that, in addition to insertions and deletions of facts, allow specify also modification requests. In tables 2 and 3, update operators are indicated with labels ι for insertions, δ for deletions and μ for modifications.

Basic Mechanism:

Two different features are considered in this dimension.

– Technique: It refers to the specific technique used by each method to propose repair actions under an integrity constraint violation.

– User Participation: It defines whether the method requires the user/designer participation during the process of obtaining solutions. This feature is important because it prevents the method to be completely automatic. We do not consider as user participation the fact that, at the end, the user may choose among several obtained solutions.

Efficiency:

Few methods take explicitly into account efficiency issues. In this sense, this dimension let us distinguish between methods that define techniques to improve the efficiency of the basic mechanism, or not. In tables 2 and 3, we indicate with *No* the methods that do not consider explicitly efficiency issues; and for each method that considers it, we indicate which issue is considered.

Soundness and ***Completeness:***

In both dimensions, we indicate for each method, if soundness and/or completeness of the method are proved (*Yes*) or not (*Not Proved*). Additionally, we indicate with label *No* the methods that provide non-valid solutions or that it do not obtain some existing correct solutions.

6. Review of Integrity Constraint Maintenance Methods

In this section, we classify and review the most relevant methods [ML91, CFPT94, Ger94, ED98, Sch98, Maa98, ST99] for integrity constraint maintenance with respect to the features identified in

Section 5. In Table 2, we summarize the result of our analysis and, in the rest of this section, we describe each method and we also comment main contributions and limitations they have.

6.1 Moerkotte and Lockemann's Method [ML91]

One of the former proposals for integrity constraint maintenance in deductive databases is the method for reactive consistency control of G. Moerkotte and C. Lockemann [ML91]. This method is completely defined at Run-time and it proposes how to obtain the repairs of an integrity constraint when the actual integrity constraint violation is detected. It distinguishes clearly three steps to obtain these repairs. In a first step, a set of symptoms is obtained from violated integrity constraints. These symptoms correspond to base and derived facts that violate some integrity constraint. In a second step, a set of causes is obtained from the symptoms. These causes correspond to base facts that raise a symptom. Finally, causes are transformed into repairs by syntactic modification.

Moerkotte's method is restricted to a particular case of databases where view predicates and integrity constraints are defined by means of rules with only positive literals. This method introduces also some techniques to improve the efficiency of the process of checking integrity constraints to identify causes. It also suggests alternative ways to help the user in the selection of the more appropriated solution of the update request.

6.2 Etzion's Method [Etz93, Etz94, ED98]

The method of O. Etzion [Etz93, Etz94, ED98] proposes the Self-Stabilization approach as an alternative to the active approach to maintain integrity constraints. At definition level, this approach is based on defining data-driven derivations (rules) instead of event-driven (active) rules. The approach is based on defining *stabilizer types*, which correspond to high-level abstractions of the repair behavior. A difference between stabilizers and active rules is that, stabilizers are more focused on the semantics of integrity constraint than the syntactic information of integrity constraints considered by active rules. However, at implementation level, stabilizer types are automatically translated to active rules.

This method allows also definition of derivation constraints, which state semantic value dependencies between derived information, by aggregation or some kind of functions, and base information.

6.3 Schewe and Thalheim's Method [ST99, Sch96, Sch00]

Another proposal to formalize integrity constraint maintenance is the work of K-D. Schewe and B. Thalheim [ST99, Sch96, Sch00]. This work deals with the problem of integrity constraint maintenance following a pure Compile-Time approach. They propose a method that, given a program specification (predefined transaction), obtains an extension of it, called greatest consistent specialization (GCS), that must ensure the initial program intention and must preserve integrity constraint satisfaction. An alternative to the GCS, the maximal consistent effect preserver (MCE) is also defined for some specific cases of program specifications. In [ST99] the authors propose a theory-based solution to the problem of integrity constraint maintenance. In [Sch00], three alternative strategies to enforce database consistency using GCS and/or MCE are proposed.

Method	Problem	Database Schema				Request	Basic Mechanism		Efficiency	Sound	Complete
	Comp / Run	Definition Language	Views	Ic definition	Type of IC	Operators	Technique	User Particip.			
ML91	Run	Logic	Yes	Limited	Static	ι δ	----	No	DB Access IC Checking	Not Proved	Not Proved
CFPT94	Comp / Run	Logic, Relational	Yes	Limited	Static	ι δ μ	Active	Yes	Ic Ordering	No	No
Ger94	Comp / Run	Logic, Relational	No	Flat, Limited	Static, Dynamic	ι δ μ	Active	Yes	Ic Ordering	No	No
ED98	Comp / Run	Logic, OO	No	Derivation, Limited	Static	ι δ μ	Self-Stabilization	Yes	Ic Ordering	Not Proved	No
Maa98	Comp / Run	Logic	No	Flat, Limited	Static, Dynamic	ι δ	Active	No	No	No	No
Sch98	Comp / Run	Logic, Relational	No	Flat, Limited	Static	ι δ	Active	No	No	Not Proved	No
ST99	Comp	Logic	Yes	Limited	Static, Dynamic	ι δ	Predefined Programs	Yes	No	Not Proved	Not Proved

Table 2. Summary of integrity constraint maintenance methods

6.4 Methods based on Active Rules [CFPT94, Ger94, Sch98, Maa98]

The active approach is one of the most frequent to maintain integrity constraints. Methods considered in this group are [CFPT94, Ger94, Sch98, Maa98].

This approach is aimed at maintaining integrity constraints through the generation at compile time of a set of active rules. When at run-time a certain transaction violates some integrity constraint, active rules are executed to guarantee that all integrity constraints remain satisfied. Active rules are generated by taking only into account the information provided by the database schema. Two types of active rules may be considered. *Condition-Action Rules* (Production Rules) specify those conditions on database contents such that when they are satisfied, an integrity constraint has been violated and, in the Action part, they specify the updates needed to repair this integrity constraint violation. *Event-Condition-Action Rules* also specify the update (event) that triggers the active rule execution when the event initiates an integrity constraint violation.

Although all these methods present different particularities regarding the generated rules or the language used to define the constraints, they share the same limitations. This is why we explain these common drawbacks together. We also comment some alternative strategies to enforce integrity constraints that these methods allow.

Integrity Constraints Definition

The most typical restriction of integrity constraint maintenance methods is that of dealing only with *flat* integrity constraints [Ger94, Maa98, Sch98]. An integrity constraint is flat if it is defined only in terms of base and/or evaluable predicates, i.e. integrity constraint definition does not contain any view literal. This is an important restriction since, as shown in the following example, not every possible condition can be defined as flat integrity constraint.

Example 6.1: Assume that we want to state that all people of working age must be employed, where people employed is defined as people that work in some company. With non-flat integrity constraints, this constraint can be defined as follows:

$$\text{Employed}(x) \leftarrow \text{Works-in}(x, y) \wedge \text{Company}(y)$$

$$\text{Ic1} \leftarrow \text{Working-age}(x) \wedge \neg\, \text{Employed}(x)$$

Notice that it is not possible to reduce integrity constraint Ic1 to a flat integrity constraint expressing the same restriction.

Some methods like [CFPT94] avoid this drawback by allowing the definition of derived information (views) in the database schema. However, the limitation they have is that, when an integrity constraint becomes violated, they are not able to repair it by requesting updates on derived information. Since they do not explicitly define any mechanism to translate the repair request (view update request) into base fact updates. The same situation appears in non-active methods like [ML91, ST99].

Soundness

As pointed out in [Sch98], the methods [CFPT94, Ger94, Maa98] may obtain solutions that do not preserve the effect of the requested update. The reason is that they do not take into account the history of database updates needed to enforce database consistency and, thus, they cannot know whether the requested update is undone by the joint effect of these updates.

Example 6.2: (adapted from [Sch98]) Assume the following database and the update request U={insert(Wire(Id1, HB, A, 2, 0))}:

Tube(Id1, HB, 4)   Wire(Id5, HB, A, 2, 0)

Wire(wire_id, conn, w_typ, volt, pwr) $\rightarrow$ Tube(tube_id, conn, t_typ)

Wire(wire_id, conn, w_typ, volt, pwr) $\wedge$ Tube(tube_id, conn, t_typ) $\rightarrow$ wire_id$\neq$tube_id

In this example, methods like [CFPT94, Ger94, Maa98] may obtain T={insert(Wire(Id1,HB,A,2,0)), delete(Tube(Id1,HB,4)), delete(Wire(Id1,HB,A,2,0)), delete(Wire(Id5,HB,A,2,0))} as solution to the update request U, but it does not satisfy the original request.

In this sense, the work of Schewe and Thalheim [Sch98, ST98] analyzes the main limitations of rule triggering systems for integrity maintenance. One of these limitations relies in the side effect preservation that appears when triggered (active) rules re-establish satisfactorily integrity constraints, but undo the initial update request. This method [Sch98, ST98] is aimed at generating active rules such that do not present this problem. Therefore, in this example, this method does not obtain the above transaction T as a solution to the initial request U.

Completeness

Methods that follow an active approach may not obtain all valid solutions that satisfy an update request. The main reason is that, they do not always consider all active rules during the integrity constraint maintenance process. In this way, they discard some potential repairs of integrity constraint violations and thus, they do not obtain all existing solutions.

For instance, [CFPT94, Ger94, Sch98] define a graph that expresses whether the execution of a certain rule that repairs an integrity constraint could violate another integrity constraint. The presence of cycles in this graph indicates that the process of integrity maintenance may not terminate. In order to guarantee termination, the database designer may remove some active rules or may define priorities among them. Therefore, at this moment completeness of the method is given up in front of termination.

Example 6.3: The same update request and database of example 6.2 may be used to show incompleteness of methods [CFPT94, Ger94, Maa98, Sch98]. Any of these methods is not able to obtain the transaction S={insert(Wire(Id1,HB,A,2,0)), delete(Tube(Id1,HB,4)), insert(Tube(Id9,HB,9))} that satisfies the initial update request and that maintains integrity constraints satisfied.

Efficiency

Methods that tackle the problem of maintaining integrity constraints are basically oriented to the procurement of all alternative ways to repair integrity constraints without putting special attention to obtain solutions efficiently. Nevertheless, there are few proposals like [CFPT94, Ger94, ED98] that explicitly consider efficiency issues.

The main cause of inefficiency during integrity constraint maintenance is that, to ensure that all integrity constraints are satisfied, each constraint can be checked and repaired several times. Concretely, each time a constraint is repaired, all integrity constraints are checked again for consistency although they were already satisfied prior to the repair and they could not become violated by the repair. This situation is illustrated in the following example.

<u>Example 6.4</u>: Assume a database that contains the following three integrity constraints Ic1, Ic2 and Ic3, and the update request to insert fact Employee(Ann).

$Ic1(p) \leftarrow Worker(p) \wedge \neg HasSalary(p)$
$Ic2(p) \leftarrow Contracted(p) \wedge \neg Worker(p)$
$Ic3(p) \leftarrow Employee(p) \wedge \neg Contracted(p)$

Consider a method that handles integrity constraints in a sequential order (i.e. Ic1, Ic2, Ic3). This method checks integrity constraints {Ic1, Ic2, Ic3} until a violation of Ic3 is detected. Ic3 is repaired by inserting Contracted(Ann). After that, integrity constraints {Ic1, Ic2} are checked again and Ic2 must be repaired with the insertion of Worker(Ann), which violates integrity constraint Ic1. It is repaired by inserting HasSalary(Ann) and the rest of integrity constraints are checked again. Finally, by checking integrity constraints in this order the method has checked eight integrity constraints.

If we take into account the interaction among repairs and possible violations of constraints, it is not difficult to see that it is enough to maintain integrity constraints in this order {Ic3, Ic2, Ic1} to obtain the previous solution. The idea is that there is an implicit order Ic3 -> Ic2 -> Ic1 to deal with these integrity constraints. The insertion of Employee(Ann) can only violate Ic3 and its repair can only violate Ic2. The repair of Ic2 can only violate Ic1 and its repair does not violate neither Ic2 nor Ic3. Finally, we check only three integrity constraints.

In order to reduce the number of times that each integrity constraint is considered, methods like [CFPT94, Ger94, ED98] define an explicit order to maintain them. Therefore, efficiency is provided by defining a graph that expresses whether the execution of a certain active rule R_i that repairs an integrity constraint Ic_j could violate another integrity constraint Ic_k.

A more elaborated technique, than the proposed in [CFPT94], to define a stratified order of handling active rules is proposed in [FP97]. However, it shares with previous work the loose of completeness because not all possible repairs are considered.

More recently, S. Jurk and M. Balaban [JB01] have proposed a technique to improve rule-triggering methods for integrity constraint maintenance. This technique tackles termination control, effect preservation and efficiency problems in a uniform way. The authors propose to use Dependency Graphs to determine an order to enforce the integrity constraints. Such order is defined: 1) to differentiate finite cyclic rule activation from non-terminating ones; 2) to prevent side effects like undoing the requested update by further integrity constraint repairs and; 3) to avoid the computational overhead of rollback operations.

Additional Approaches to Integrity Constraint Enforcement

Some of the analyzed methods in this section are not restricted to follow an integrity constraint maintenance approach to enforce integrity constraints. Methods like [Ger94, ED98, Maa98] are able to apply different strategies to enforce integrity constraints.

The method proposed by M. Gertz [Ger94] allows to explicitly specify what kind of reaction to apply depending on the update that violates an integrity constraint. In addition to the maintenance approach, this method deals with partial rollbacks of the violating updates instead of all the transaction; it allows to modify the transaction updates that produce the violation; and it also may accept violated integrity constraint as exceptions.

The method of S. Maabout [BM97, Maa98] does not require that in the current state all integrity constraints are satisfied. Therefore, the method could be used indistinctly to maintain integrity constraints or to restore integrity constraints depending on whether in the current state, integrity constraints are satisfied or not.

7. Relevant Features for View Updating

In this section, we extend the features identified in Section 5 to classify integrity constraint maintenance methods with some specific features relative to the view updating problem. The additional features of each dimension we take into account are the following:

Database Schema:

– Views: Obviously, all methods that tackle the problem of view updating must allow the definition of views or derived information. However, some methods may impose restrictions on the views they handle. In Table 3, we explicitly indicate those restrictions. Methods with label *Yes* do not restrict view definitions.

Basic Mechanism:

Two additional features are considered in this dimension.

– Base Facts: When an update request is translated into a set of transactions, it is necessary to consider the current contents of the database. If a method does not consider it, the method could be inefficient, incomplete and/or it could obtain redundant solutions.
– Loop Control: Update requests on recursive views may cause a method to enter in an infinite loop. In Table 3, we distinguish those methods that define a mechanism to handle updates on recursive views (*Yes*) from those that do not define it (*No*) or that do not handle recursive views (*n.a.*).

Solutions:

A requirement that many methods impose to solutions of a view update request is that they must be minimal.

– Minimal solutions: When several alternative solutions exist, methods use to define a criterion to obtain the more representative ones. A common criterion is that solutions must be minimal, but not all methods consider the same definition of minimal solution. In Table 3, we indicate the criterion of minimality that each method considers, and whether the obtained solutions fulfill (or not) this requirement.

8. Survey on Methods for View Updating and Integrity Constraint Maintenance

Methods dealing with view updating and integrity constraints maintenance that we have considered in this survey are [KM90, Wüt93, CHM95, CST95, PO95, TO95, Dec97, LT97, IS99, MT00]. In Table 3, we summarize the analysis of these methods with respect to the features identified in sections 5 and 7.

We have classified the above methods in two different groups depending on the applied technique. The first group includes methods that incorporate the information provided by the integrity constraints into the update request and then unfold the resulting expression [Wüt93, CST95, LT97]. The second group includes methods that take into account the integrity constraints every time that a new update is considered [KM90, CHM95, PO95, TO95, Dec97, IS99, MT00]. These groups are described and analyzed in more detail, respectively, in Sections 8.1 and 8.2.

We have considered also two methods for view updating [LLS93, AB99]. They have been included in Table 3, but we have not described them in more detail since they are not able to handle integrity constraints.

8.1 Methods that Extend the Update Request

Methods of this group [Wüt93, CST95, LT97] distinguish clearly two steps to obtain solutions to an update request. Given an update request U, the first step is aimed at obtaining a formula F, defined only in terms of base predicates, that characterizes all solutions to the update request. This formula is obtained by incorporating information of integrity constraints in U and by unfolding derived predicates by their corresponding view definitions until no more unfolding can be performed. In the second step, the obtained formula is analyzed to determine the insertions and deletions of base facts that satisfy the update request and do not violate integrity constraints. Methods that follow this approach differ in how they integrate information of integrity constraints into the update request U.

Method	Problem	Database Schema				Request	Basic Mechanism				Efficiency	Minimal Solution	Sound	Complete
	Comp / Run	Definition Language	Views	Ic definition	Kind of IC	Operator	Technique	User Part.	Base Facts	Loop Control				
KM90	Run	Logic	Yes	Yes	Static	ι δ	Abduction (SLDNF)	No	No	No	No	Subset No	No	No
LLS93	Run	Logic	Yes	n. a.	n. a.	ι δ	---	No	Yes	No	No	Yes	No	---
Wüt93	Run	Logic	Yes	Yes	Static	ι δ	Unfolding	Yes	No	Yes	DB Access	Subset No	Not Proved	No
CHM95	Comp / Run	OO	Class, Attribute	Limited	Static	ι δ	Active	No	Yes	n.a.	No	Not Required	Yes	No
CST95	Run	Logic	Yes	Flat, Limited	Static	ι δ	Abduction (Unfolding)	No	Yes	Yes	No	Subset No	Yes	Not Proved
PO95	Compile	Logic	Yes	Yes	Static, Dynamic	ι δ	Program Synthesis	No	n. a.	No	No	Subset No	Yes	No
TO95	Comp / Run	Logic	Yes	Yes	Static, Dynamic	ι δ	SLDNF	No	Yes	No	No	Subset Yes	Yes	Yes
Dec97	Run	Logic	Yes	Yes	Static	ι δ	Abduction (SLDNF)	No	No	No	No	Not Required	Not Proved	No
LT97	Run	Logic	Yes	Flat, Limited	Static	ι δ	Unfolding	Yes	Yes	Yes	No	P-order Yes	No	Not Proved
IS99	Run	Logic	Acyclic Covered	Limited	Static	ι δ	Abduction (Fixpoint)	No	Yes	n.a.	No	Subset Yes	Yes	Yes
AB00	Run	Logic	Ground Definite	n. a.	n. a.	δ	Hyper-Tableau	No	Yes	n. a.	No	Hitting Set Yes	Yes	Yes
MT00	Comp / Run	Logic	Yes	Yes	Static, Dynamic	ι δ μ	SLDNF	No	Yes	No	Ic Ordering View Updating	Subset Yes	Yes	Yes

Table 3. Summary of view updating and integrity constraint maintenance methods

<u>Wüthrich's Method [Wüt93]</u>

This method characterizes an update request as a conjunction of insertions and deletions of base and/or derived facts. A solution is then characterized by the set of base facts to be inserted (I) and base facts to be deleted (D) from the extensional database in order to satisfy the requested update and the integrity constraints.

The general approach to draw the solutions follows the two step approach outlined before. The first step is completely automatic. In this step, integrity constraints are incorporated into the update request. Moreover, the unfolding process makes an explicit control when recursive views are involved. In the second step, the solutions (transactions) are obtained. When alternative ways to generate a solution exist, the user is requested to solve possible ambiguities.

This method has two main limitations: it is not complete and it does not necessarily generate minimal solutions.

<u>Completeness</u>

Wüthrich's method implicitly assumes that there is an ordering for dealing with the deductive rules and integrity constraints involved in the update request that will lead to the generation of a solution. However, this ordering does not always exist. The following example shows that this assumption may impede this method to obtain all valid solutions.

<u>Example 8.1</u>: Given the database:

Node (A) Node (B) Edge (A, B) Edge (B, A)

$Ic1 \leftarrow Node\ (x) \wedge \neg\ \exists y\ Edge\ (x, y)$ $Ic2 \leftarrow Node\ (x) \wedge \neg\ \exists z\ Edge\ (z, x)$

$Ic3 \leftarrow Edge\ (x, y) \wedge \neg\ Node\ (x)$ $Ic4 \leftarrow Edge\ (x, y) \wedge \neg\ Node\ (y)$

and the update request insert(Edge(A,C)), Wüthrich's method could not obtain the solution characterized by the sets I={Edge(A,C), Node(C), Edge(C,D), Node(D), Edge(D,B)} and D=∅. This problem will mainly appear when the knowledge base contains referential integrity constraints, such as the above ones.

<u>Minimal Solutions</u>

Wüthrich's method is defined to obtain only minimal solutions. A solution S is not minimal if there is a subset of S that it is also a solution. However, this method does not necessarily generate minimal solutions because it does not check whether a base or view fact is already present in the database when suggesting to insert or delete it. This is shown in the following example.

<u>Example 8.2</u>: Given the database:

S(A, B) $P(x) \leftarrow Q(x) \wedge R(x)$ $R(x) \leftarrow S(x, y)$

Given the request insert(P(A)), Wüthrich's method could only obtain the non-minimal solution I={Q(A), S(A,C)}, where C is a value given by the user or assigned by default. There exists another solution I={Q(A)} which is a subset of the previous one that is not obtained by this method.

Console, Sapino and Theseider's Method [CST95]

The method proposed in [CST95] follows an abductive approach. The two steps of this method proceed as follows. In the first step, it is obtained a formula F* in disjunctive normal form by considering the update request ϕ and the integrity constraints. The update request ϕ is unfolded until a formula F without derived literals is obtained. Afterwards, F* is obtained by incorporating into F the syntactical residues of integrity constraints potentially violated by literals of F.

In the second step, F* is instantiated and simplified by considering the contents of the extensional database and values given by the user. At the end, a ground formula in disjunctive normal form F** is obtained characterizing all solutions to the update request.

This method presents two main limitations: it only deals with restricted integrity constraints and in some cases, it can obtain solutions that are not minimal.

Integrity Constraints Definition

As shown in Table 3, the method can only handle flat integrity constraints. We already have commented in section 6.1 how this limitation restricts the expressive power of integrity constraints. Moreover, this limitation is more significant in methods that deal with view updates than in methods that do not.

However, this method considers only two particular types of flat integrity constraints:

– Integrity constraints defined as $\neg(P(x) \wedge Q(y))$ where P and Q are base predicates.

– Non-cyclic referential integrity constraints of the form $P(x) \rightarrow \exists y\, Q(x,y)$.

The above restrictions limit considerably the expressive power of integrity constraints since they are not able to specify restrictions that are usually find in real-world situations.

Minimal Solutions

In some cases, the formula F** may characterize solutions that are not minimal. This situation occurs when F** has disjunctands that are subsumed by other disjunctands.

Example 8.3: Consider the update request $\phi = \neg P$ to delete the view fact P in the following database:

R(1) R(2) S(2)

$P \leftarrow R(x) \wedge \neg S(x)$

The formula we obtain is $F^{**} = [\neg R(1) \wedge \neg R(2)] \vee [\neg R(1)] \vee [S(1) \wedge \neg R(2)] \vee [S(1)]$ where each disjunctand characterize a correct solution. Notice that the first and third disjuncts do not characterize minimal solutions because literal $\neg R(2)$ is unnecessary to satisfy the update request ϕ. In this example, minimal solutions are T1={delete(R(1))} and T2={insert(S(1))}.

Lobo and Trajcevsky's Method [LT97]

In a similar way, this method unfolds the update request to obtain also a disjunctive normal formula F. This formula is then extended with residues of the integrity constraints potentially violated by F. At the end, non-ground variables are instantiated by considering facts of the extensional database. The method also analyzes and formalizes the process of how to give values to existential variables.

Given an update request, the purpose of the method is to obtain only one solution that must be minimal. The criterion of minimal solution is not based on the set inclusion as other methods do. They define a *Prevention order* (P-order) between solutions. In the sense, given two solutions, the minimal one is that has few deletions, and in case both solutions have the same number of deletions, the minimal solution is such that it has few insertions.

This method presents two different drawbacks:

Integrity Constraints Definition

This method also restricts the integrity constraints to be flat. Moreover, integrity constraints must be *resolution complete*. That is, it must not be possible to derive new (implicit) integrity constraints from given set of integrity constraints. For instance, $Ic1 \leftarrow Q(x) \wedge \neg R(x)$ and $Ic2 \leftarrow R(x) \wedge S(x)$ are not resolution complete since a third integrity constraint can be deduced from them: $Ic3 \leftarrow Q(x) \wedge S(x)$. The problem is that, as far as we know, there is no mechanism to derive sets of integrity constraints that are resolution complete.

Soundness

This method is not always correct since the formula F does not always characterize valid solutions, as shown in the following example.

Example 8.4: Given the update request insert(Q(B,2)) to apply to the database:

$S(A, 1)$ $\qquad Q(x, y) \leftarrow \neg P \wedge S(x, y)$ $\qquad P \leftarrow S(x, y) \wedge \neg T(y)$

this method would obtain two minimal solutions: T1={delete(S(A,1)), insert(S(B,2))} and T2={insert(T(1)), insert(S(B,2))}. However, none of them satisfies the requested update since insert(S(B,2)) induces P and, hence, it falsifies insert(Q(B,2)).

Moreover, there exist two correct solutions: S1={delete(S(A,1)), insert(S(B,2)), insert(T(2))} and S2={insert(T(1)), insert(S(B,2)), insert(T(2))} that are not obtained by this method.

8.2 Methods that Consider Integrity Constraints Dynamically

The methods analyzed in this second group are [KM90, CHM95, PO95, TO95, Dec97, IS99, MT00]. These methods take into account integrity constraints every time a new update is proposed to be included in the solution. Therefore, only those integrity constraints that could be violated by the proposed update are taken into account.

The mechanisms used by these methods to generate solutions of an update request are different. Some of them [KM90, TO95, Dec97, MT00] are based on extensions of SLD or SLDNF proof procedure, [CHM95] follows an active approach, while [IS99] is based on the computation of the fixpoint of a program. The method [PO95] follows a compile approach and it is based on the generation of programs specifications.

Kakas and Mancarella's Method [KM90]

This method follows an abductive approach to translate view update requests and maintain integrity constraint consistency. It clearly defines two steps: in the first one, the update request (U) is translated into different sets (Δ_i). Each set Δ_i specifies the requirements of the extensional database that should be fulfilled to satisfy the update request U. Therefore, each Δ_i must provide a correct solution to the update request U. In the second step, each set Δ_i is used to obtain a transaction to update the extensional database.

This method has two limitations.

Unnecessary work

The first problem of the method relies on the fact that, during the first step, extensional database is not explicitly considered when requirements of sets Δ_i are obtained. Therefore, a set Δ_i could define some requirement that has already been satisfied by the current contents of the EDB. In this sense, at the end of the first step, some of the obtained sets Δ_i have redundant requirements that are not useful to the generation of transactions in the second step. This is shown in the following example.

Example 8.5: Consider the update request to delete derived fact P from this database:

$Q \qquad R$
$P \leftarrow Q \wedge R \qquad\qquad P \leftarrow S \wedge T$
$S \leftarrow A \wedge B \qquad\qquad T \leftarrow C \wedge D$

At the end of the first step, this method obtains the following sets of database requirements:

$\Delta_1 = \{\neg P, \neg Q, \neg A\} \qquad \Delta_5 = \{\neg P, \neg R, \neg A\}$
$\Delta_2 = \{\neg P, \neg Q, \neg B\} \qquad \Delta_6 = \{\neg P, \neg R, \neg B\}$
$\Delta_3 = \{\neg P, \neg Q, \neg C\} \qquad \Delta_7 = \{\neg P, \neg R, \neg C\}$
$\Delta_4 = \{\neg P, \neg Q, \neg D\} \qquad \Delta_8 = \{\neg P, \neg R, \neg D\}$

Notice that requirements $\neg A, \neg B, \neg C, \neg D$ are satisfied by the current database contents. Therefore, instead of generating and evaluating the above eight sets, it is enough to define the sets $\Delta_{10} = \{\neg P, \neg Q\}$ and $\Delta_{11} = \{\neg P, \neg R\}$ to obtain, in the second step, transactions T_1 = {delete(Q)} and T_1 = {delete(R)}.

Soundness

In some cases, this method does not detect all integrity constraint violations.

Example 8.6: Consider the update request of inserting base fact S(A) in the following database:

$S(x) \leftarrow Q(x)$
$P(x) \leftarrow Q(x)$
$Ic \leftarrow P(x) \wedge \neg R(x)$

At the end of the firs step, the method obtains only one set of requirements Δ = {Q(A)}. This requirement can be satisfied in the second step by the transaction T={insert(Q(A)}. Notice that, although this transaction satisfies the update request of inserting S(A), the integrity constraint remains violated. Therefore, this transaction can not be considered a valid solution to the update request.

Chen, Hull and Mcleod's Method [CHM95]

This method is based on the execution of active rules to update derived data in a semantic object-oriented database model. It follows an integrity constraint checking approach to enforce integrity constraints, although for a few types of integrity constraints it proposes how to repair an integrity constraint violation.

Active rules are described by means of *Limited Ambiguity Rules (LAR*), a kind of condition-action rules that are automatically obtained from the database schema at compile time. Two kinds of LAR rules are considered: the upward rules that propagate changes from base classes to derived classes, while the downward rules propagate changes in the opposite direction.

At run time, given an update request (Δ), LAR rules are executed according to the *Principle of Down-Up Propagation*, which determines the execution order of LAR rules. The main idea is that downward rules must be always executed before upward rules. The execution of LAR rules corresponds to a breath-first search of all alternative completions of the initial update request Δ that will satisfy Δ and all integrity constraints. Therefore, all solutions to Δ are obtained.

The derived information is restricted to derived attributes and classes. Derived attributes are defined by means of arithmetic or aggregate expressions. Derived classes are restricted to be subclasses or defined as the union/difference/intersection of other classes. Recursive class/attribute definitions are not allowed.

Integrity constraint definition is very restrictive. This method can only handle structural integrity constraints (type domains, ISA hierarchies) and user-defined integrity constraints of attributes and their values (cardinality, no overlapping values, inclusion, …). It follows an integrity constraint checking strategy. Nevertheless, it maintains constraints on univalued attributes, on no-overlapping values and on implicit range/domain attributes by including the repairing actions in the corresponding LAR definition.

This method requires solutions to be minimal, with respect the set inclusion criterion. Moreover, a solution can not include the insertion and deletion of the same fact. Both conditions are checked after the execution of all the upward and downward rules. Therefore, some of the obtained completions do not correspond to actual solutions. In this sense, this method can perform some unnecessary work due to the fact that the above conditions are not checked as soon as possible.

The Events Method [TO95]

The Events Method models updates by means of events, which are used to generate transition and event rules. Transition rules defines the contents of the new state of the database in terms of the old database state and the updates, while event rules define the events induced during the transition to the new state. Both sets of rules, together with the original database, form the Augmented Database[Oli91].

The Events Method is based on an extension of SLDNF and it uses the Augmented Database to obtain the solutions that satisfy a request.

Given an update request u and the Augmented Database A(D), the Events Method obtains all minimal solutions T_i to u. Each T_i is obtained by having some failed SLDNF derivation of $A(D) \cup \{ \leftarrow u \wedge \neg\iota Ic$[1]$\}$ succeed. This is achieved by including in the transaction T_i each positive base event fact (base update) selected during the failed derivation. At the end, we have that there is an SLDNF refutation of $\leftarrow u \wedge \neg\iota Ic$ by considering $A(D) \cup T_i$ as input set. Different ways to make failed derivations succeed, correspond to different solutions of u.

The main contribution of the Events Method is to be sound and complete. That is, all obtained solutions satisfy the update request and do not violate any integrity constraint, and it obtains all possible solutions. Moreover, this method can also be used to maintain dynamic integrity constraints, to insert and delete views and integrity constraints, to request qualified updates and to prevent side effects on other views.

The main drawback of the Events Method is that it does not consider efficiency issues. Therefore, trying to obtain all solutions of an update request, this method explores all branches of the associated SLDNF derivation tree and, for each obtained solution all integrity constraint may be checked, and possibly repaired, several times. Moreover, if the update is requested on an existential view, this method must consider all possible instantiations of a given variable. This method may also enter in an infinite loop trying to obtain all solutions of an update request on a recursive view since it does not keep under control these situations.

Pastor and Olivé's Method [PO95]

This work proposes a method for view updating and integrity constraint maintenance following a compile time approach. Given a predefined update request specified by the designer, this method automatically derives a program specification. At run time, formal parameters of the program are instantiated to actual values and the program is executed. Execution of this program generates a transaction that satisfies the update request without violating any integrity constraint.

The mechanism this method uses to synthesize a program specification is based on the use of the Augmented Database like the Events Method [TO95]. However, since this method follows a compile time approach, base facts stored in the database and actual values of the update request are not available during the program synthesis.

The main contribution of this work is to be the first compile-time proposal that tackles, in an integrated way, view updating and integrity constraint maintenance in deductive databases. Moreover, this method is

[1] Literal $\neg\iota Ic$ corresponds to the prevention of violating any integrity constraint.

sound in the sense that transactions obtained by execution of a program specification always satisfy the user request without violating integrity constraints.

The main limitations of this method are that it is not complete and that it may generate non-minimal solutions. In some cases, the execution of a program specification could not be able to provide some alternative correct transaction and it could provide transactions containing base fact updates that are not strictly necessary to satisfy the update request. This is because the actual update request and database contents are not available during the program synthesis.

Decker's Method [Dec96, Dec97]

This method is based on the SLDAI resolution procedure, which is an abductive extension of the SLD resolution procedure. The SLDAI procedure is an interleaving of *refutation* and *consistency* derivations. Given an update request, the refutation derivation pursues the empty clause by considering the database contents. During this derivation, new hypotheses are included in the solution set H. Every time a hypothesis is included in H, its consistency is verified by a consistency derivation.

The main limitation of this method is that it cannot manage appropriately update requests that involve rules with existential variables. The reason is that refutation derivations flounder when a literal corresponding to a non-ground base predicate is selected, thus impeding to reach the empty clause.

Example 8.7: Given the update request to insert P into a database with only one rule $P \leftarrow S(x)$, the Decker's method obtain the following refutation:

$$\begin{tikzcd}
\leftarrow P \arrow[d, dash] \\
\leftarrow S(x) \arrow[d, dash] \\
\text{flounders}
\end{tikzcd}$$

In this case, the method flounders and it is not able to obtain any solution even though there are as many correct solutions as possible values of x for which to insert S(x).

The same problem appears in the consistency derivations, because this method does not take into account base facts during the consistency derivations.

Example 8.8: Consider the update request to insert derived fact P in the following database:

$R(A, B) \qquad P \leftarrow Q(A) \qquad \leftarrow Q(x) \wedge R(x, y) \wedge \neg S(y)$

The main refutation and consistency derivations of the update request are the following:

$$\begin{tikzcd}
\text{REF.} & \text{CONS.} & \\
\leftarrow P \arrow[d, dash] & Q(A) \leftarrow \arrow[d, dash] & H=\{Q(A)\} \\
\leftarrow Q(A) \arrow[d, dash] & \leftarrow R(A, y) \wedge \neg S(y) \arrow[d, dash] & (*) \\
\text{flounders} & \text{flounders} &
\end{tikzcd}$$

In the derivation step (*), both literals are not ground. Since the method does not take into account extensional database, it can not be resolved by considering the base fact R(A, B). Therefore, the method flounders and does not obtain any solution to the update request. Notice, however, that there are two minimal solutions to the update request T1={insert(Q(A)), delete(R(A,B))} and T2={insert(Q(A)), insert(S(B))}.

Inoue and Sakama Method [IS99]

Inoue and Sakama propose a method that follows an abductive approach, in which two steps are differentiated. Given an abductive program <P, A>, where P is an acyclic normal logic program (with or without integrity constraints) and A is the set of abducible atoms, in the first step, an automatic transformation is applied to P to obtain a transaction program τP of P with respect to A. A transaction program τP specifies in a declarative way requests to add/delete hypotheses to/from P by means of literals *in(A)*/*out(A)*, respectively.

In the second step, to obtain explanations of an observation G, the method computes a fixpoint of the transaction program τP ∪ G in a bottom-up manner. As a result, several minimal explanations <E, F> can be obtained. The set E specifies the hypothesis (abducible atoms) to add to program P, while the set F specifies the hypothesis (abducible atoms) to be removed from P.

Relating this approach to the view updating and integrity maintenance problems, an observation G corresponds to the update request, abductive atoms correspond to base predicates, program P corresponds to the database and minimal explanations <E,F> correspond to minimal solutions.

This method is sound and complete for covered and acyclic programs. However, this method has two important drawbacks. The first one relies in the fact that this method can not deal with programs with existential rules. The second one is related to efficiency, since to obtain valid explanations it may perform some unnecessary work.

Existential case

This method can only be applied to *covered* normal logic programs. That is, for every rule of program P, all variables that appear in the body of a deductive rule, must also appear in its head.

Integrity constraints are defined by rules with an empty head (⊥). Therefore, this method can only handle ground integrity constraints. Notice that this restriction limits considerably the integrity constraints that this method can deal with. For example, this method can not handle integrity constraints like:

$\perp \leftarrow \text{Professor}(p) \wedge \neg\, \text{Doctor}(p)$

$\perp \leftarrow P(k, x) \wedge P(k, y) \wedge x \neq y$

In a similar way, definition of derived or view predicates is also restrictive. For example, view predicates defined as projections can not be defined and thus handled by this method:

$\text{Contracted}(p) \leftarrow \text{Has-Contract-in}(p, c)$

Unnecessary Work

All obtained solutions to an update request G must be *coherent*. That is, a solution can not add and delete the same base fact. That is, for each solution <E, F> sets E and F must be disjoint ($E \cap F = \emptyset$). Moreover, solutions must be also *minimal*. That is, a solution <E', F'> is not minimal if there is another solution <E, F> ($E \neq E'$ and $F \neq F'$) such that $E \subseteq E'$ and $F \subseteq F'$.

The coherency and minimality conditions of the obtained solutions are not checked during the process of computing the fixpoint. They are checked after obtaining the fixpoint. Therefore, the obtained fixpoint may contain transactions that should be rejected because they can not provide coherent and minimal solutions. In this sense, this situation may cause to perform some unnecessary work. In the following example, we show this extra work when integrity constraints are not affected by the requested update G.

Example 8.10: Consider the update request of inserting base fact B in the following deductive database where base predicates are W, S, C, E and B:

$\bot \leftarrow W \wedge \neg S$ $\quad\quad$ $\bot \leftarrow C \wedge \neg W$ $\quad\quad$ $\bot \leftarrow E \wedge \neg C$

The obtained transaction program (τP) of P with respect to A is the following:

$out(\bot) \rightarrow out(W) \mid in(S)$ $\quad$ $out(W) \rightarrow \varepsilon$ $\quad$ $out(S) \rightarrow \varepsilon$
$out(\bot) \rightarrow out(C) \mid in(W)$ $\quad$ $out(C) \rightarrow \varepsilon$ $\quad$ $out(E) \rightarrow \varepsilon$
$out(\bot) \rightarrow out(E) \mid in(C)$ $\quad$ $out(B) \rightarrow \varepsilon$

The fixpoint obtained of the update request to insert B without violating any integrity constraint G = {in(B), out(⊥)} is the following:

T={{in(B)}, $\quad$ {in(B), in(S)},
~~{in(B), out(W), out(C), in(C)}~~, $\quad$ ~~{in(B), in(S), out(C), in(C)}~~,
~~{in(B), out(W), in(W), out(E)}~~, $\quad$ {in(B), in(S), in(W)},
~~{in(B), out(W), in(W), in(C)}~~, $\quad$ {in(B), in(S), in(W), in(C)}}

Notice that in this case, the insertion of fact B does not violate any integrity constraint. However, the fixpoint computation obtains eight alternative transactions. Four of them are not coherent but only the first one is minimal. Therefore, the unique minimal solution of the update request G is $S^o = (\{B\}, \emptyset)$.

Mayol and Teniente's Method [MT99a, MT00]

The method proposed by E. Mayol and E. Teniente is an extension of the Events Method [TO95] and it is also sound and complete for hierarchical databases.

The main contributions of this method rely on the update operators considered and on incorporating specific techniques to improve efficiency of view updating and integrity constraint maintenance.

As far as the update operators are concerned, this method considers the modification as a single update operator both at the derived and base fact definition levels. By considering modifications explicitly, key integrity constraints are implicitly enforced by the own definition of this method.

Efficiency is provided at two different moments. On the one hand, this method defines a graph, the Precedence Graph, which explicitly defines the relationships between repairs and potential violations of integrity constraints. Then, the Precedence Graph allows this method to deal with the integrity constraints into a proper order to reduce the number of times each integrity constraint must checked and repaired. On the other hand, this method analyses the view update request to discard all potential ways to translate it that do not lead to a correct translation. These techniques are described in detail in [MT99a].

The main drawbacks of this method are dealing with existential rules, since in general either all possible instantiations of a given variable must be considered (which is clearly unpractical) or this instantiation is requested to the user (which may lead to not obtain some valid solution). Moreover, this method does not deal with recursive views properly, since it may loop infinitely trying to obtain some solution.

9. Conclusions and Further Work

We have reviewed in detail the most relevant methods that deal with integrity constraint maintenance and with view updating.

In this sense, we have defined a general framework to classify and to compare these methods. As a relevant dimensions of this framework, we have considered: the *problem* each method tackles; the *database* schema it can handle; what kind of *update operators* a method manages; the basic *mechanism* that defines the method; the specific techniques used to improve the *efficiency*; and, finally, the solutions each method obtains evaluating its *soundness* and *completeness*. Moreover, for the analyzed methods, we have described the approach they follow, their contributions and the main limitations they may have.

Taking into account the problem that each method tackles, we have distinguished two groups of methods. The first group comprises integrity constraint maintenance methods like [ML91, CFPT94, Ger94, ED98, Maa98, Sch98, ST99]. The second group of methods includes methods like [KM90, Wüt93, CHM95, CST95, PO95, TO95, Dec97, LT97, IS99, MT00] that tackle the integrity constraints maintenance and view updating problems in an integrated way.

After our analysis, we state that the main limitations of these methods are related to the kind of integrity constraints they can handle; if they can obtain all correct solutions of an update request and, that in general, efficiency issues are not always taken into account.

With respect to the *integrity constraint definition*, we have noticed that methods like [Ger94, CST95, LT97, ED98, Maa98, Sch98, ST99] can only handle flat integrity constraints. Moreover, some methods impose additional restrictions to the integrity constraint definition.

With respect to solutions obtained to satisfy an update request, we have noticed three main lacks. First one refers to the *minimality of solutions*. Some view updating methods [KM90, Wüt93, CST95, PO95] may provide solutions that do not satisfy the requirement to be minimal. The second one is related to the *completeness* of the method. In some cases, methods like [KM90, Wüt93, CFPT94, Ger94, CHM95, PO95, Dec97, ED98, Maa98, Sch98] can not obtain correct solutions that other methods obtain. The third drawback refers to *soundness*. In some cases, methods like [KM90, CFPT94, Ger94, LT97, Maa98, IS99] obtain incorrect solutions. They are solutions that violate some integrity constraint or that, although they satisfy all integrity constraint, they do not satisfy the requested update.

One important open research topic in this field is efficiency of the process to translate view updates and maintain integrity constraints. This situation contrasts with the research on the integrity constraint checking approach, where efficiency is the main topic of interest. There are few proposals that define specific techniques to translate view updates or to perform database accesses in an intelligent and efficient way [Wüt93, ML91, MT00]. Moreover, there are also very few proposals that consider efficiency issues during the process of maintaining integrity constraints [CFPT94, Ger94, ED98, MT00]. In general, they are based on the definition of a proper order to repair integrity constraints reducing, in this way, the number of times each integrity constraint must be checked and repaired.

Although we have encountered several promising proposals, we may conclude after our analysis, there is a need for further research in view updating and integrity maintenance. As far as effectiveness is concerned, there is no method yet that handles recursive rules satisfactorily. Moreover, with respect to efficiency issues, the current methods are not able to handle efficiently view updates though deductive rules with existential variable. We believe that these limitations should be overcome, at least the second one, if we want to incorporate the treatment of these problems into commercial database systems.

Acknowledgements

The description and performance of the methods considered in this survey are based on our understanding of the related papers, hence any error is entirely our own. However, we would like to thank authors of most of these methods for their comments about our analysis.

We are grateful also to D. Costal, C. Farré, A. Olivé, J. A. Pastor, C. Quer, M. R. Sancho, J. Sistac and T. Urpí for many useful comments and discussions.

References

[AB00] Aravindan, Ch.; Baumgartner, P. “Theorem Proving Techniques for View Deletion in Databases“ Journal of Symbolic Computation, Vol. 29(2), Feb. 2000, pp. 119-147.

[Ban79] Bancilhon, F. “Supporting View Updates in Relational Data Bases“ In Data Base Architecture (Bracci and Nijssen Eds.), North Holland, Amsterdam, 1979.

[BM97] Bidoit, N.; Maabout, S. “A Model Theoretic Approach to Update Rule Programs“ Proc. of the Int. Conference of Database Technology ICDT’97, Delphi, Greece, January, 97, LNCS-1186, pp. 173-187.

[BS81] Bancilhon, F.; Spyratos, N. “Update Semantics of Relational Views“ ACM Transactions on Database Systems Vol.6(4), Dec. 1981, pp.557-575.

[BV88] Brosda, V.; Vossen, G. “Update and Retrieval in a Relational Database through a Universal Schema Interface“ ACM Transactions on Database Systems Vol.13(4), Dec. 1988, pp.449-485.

[CA79] Carlson, C.R.; Arora, A.K. “The Updatability of Relational Views Based on Functional Dependencies“ Proc. of the 3rd International Computer Software and Applications Conference (COMPSAC’79), IEEE Computer Society, Chicago,1979, pp. 415-420.

[CFPT94] Ceri, S.; Fraternali, P.; Paraboschi, S.; Tanca, L. "Automatic Generation of Production Rules for Integrity Maintenance" ACM Transactions on Database Systems Vol.19(3), Sept. 1994, pp.367-422.

[CGMD94] Celma, M.; García, C; Mota, L.; Decker, H. “Comparing and Synthesizing Integrity Checking Methods for Deductive Databases”, 10th Int. Conf. on Data Engineering (ICDE), Houston, USA, 1994, pp. 214-222.

[CHM95] Chen, I.A.; Hull, R.; McLeod, D. "An Execution Model for Limited Ambiguity Rules and Its Application to Derived Data Update", ACM Transactions on Database Systems, Vol. 20(4), Dec. 1995, pp. 365-413.

[CP84] Cosmadakis, S.; Papadimitriou, C. “Updates in Relational Views“ Journal of the Association for Computer Machinery Vol.31(4), Oct. 1984, pp. 742-760.

[CST95]Console, L.; Sapino, M.L.; Theseider,D. "The Role of Abduction in Database View Updating", Journal of Intelligent Information Systems, Vol. 4(3), 1995, pp.261-280.

[Dat86] Date, C.J. “Updating Views“ in Relational Databases: Selected Writings, Addison Wesley, 1986, pp. 367-395.

[Dec96] Decker, H. "An Extension of SLD by Abduction and Integrity Maintenance for View Updating in Deductive Databases", Joint Int. Conference and Symposium on Logic Programming, September 1996, Bonn, pp.157-169.

[Dec97] Decker, H. " One Abductive Logic Programming Procedure for two kind of Updates", Proc. Workshop DINAMICS'97 at Int. Logic Programming Symposium (ILPS), September 1997, Port Jefferson, N.Y.

[ED98] Etzion, O.; Dahav, B. "Patterns of self-stabilization in database consistency maintenance", Data & Knowledge Engineering, Vol. 28(3), 1998, pp.299-319.

[Etz93] Etzion, O. "A Reflective Approach for Data-Driven Rules", Int. Journal of Intelligent and Cooperative Information Systems, Vol. 2(4), 1993, pp.399-424.

[Etz94] Etzion, O. "An Alternative Paradigm for Active Databases", Fourth Int. Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, Texas, February 1994, pp.39-45.

[FC85] Furtado, A.L.; Casanova, M.A. “Updating Relational Views“ in Query Processing in Database Systems (W. Kim et al. Eds.), Springer-Verlag, 1985, pp. 127-142.

[FP93] Fraternali, P.; Paraboschi, S. "A Review of Repairing Techniques for Integrity Maintenance" First Int. Workshop on Rules in Database Systems (RIDS'93), Edinburg 1993, pp. 333-346.

[FP97] Fraternali, P.; Paraboschi, S. "Ordering and Selecting Production Rules for Constraint Maintenance: Complexity and Heuristic Solution" Transactions on Knowledge and Data Engineering, Vol. 9(1), 1997, pp. 173-178.

[FSS79] Furtado, A.L.; Sevcik, K.C.; dos Santos, C.S. "Permitting Updates through Views of Databases", Information Systems, Vol. 4(4), 1979, pp.269-283.

[GA93] Grefen, P.; Apers, P. "Integrity Control in Relational Database Systems: An Overview" Data & Knowledge Engineering, Vol. 10, 1993, pp. 187-223.

[Ger94] Gertz, M. "Specifying Reactive Integrity Control for Active Databases", Proceedings of RIDE'94, Houston, Texas, 1994, pp. 62-70.

[IS99] Inoue, K.; Sakama, Ch. "Computing Extended Abduction through Transaction Programs" Annals of Mathematics and Artificial Intelligence, Vol. 25(3-4), 1999, pp. 339-367.

[JB01] Junk, S.; Balaban, M. "Improving Integrity Constraint Enforcement by Extended Rules and Dependency Graphs", 12th International Conference on Database and Expert Systems Applications (DEXA’01), Munich, Germany, September 2001, pp. 501-516.

[Kel85] Keller, A.M. "Algorithms for Translating View Updates to Database Updates for Views Involving Selection, Projections and Joins", Proc. of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS'85), Portland, Oregon, 1985, pp. 154-163.

[Kel86] Keller, A.M. "The Role of Semantics in Translating View Updates", IEEE Computer, Vol. 19(1), January 1986, pp. 63-73.

[Kel87] Keller, A.M. "Comments on Bancilhon and Spyratos: 'Update Semantics of Relational Views'", Technical Note, ACM Transactions on Database Systems Vol. 12(3), Sept. 1987, pp.521-523.

[KKT92] Kakas, A.C.; Kowalsky, R.A.; Toni, F. "Abductive Logic Programming", Journal of Logic and Computation, Vol 2, 1992, pp. 719-770.

[KM90] Kakas, A.C.; Mancarella,P. "Database Updates Through Abduction", Proc. of the 16th VLDB Conference, Brisbane, Australia, 1990, pp. 650-661.

[KU84] Keller, A.M.; Ullman, J.D. "On Complementary and Independent Mappings", Proc. of ACM SIGMOD International Conference of Data, Boston, SIGMOD Record Vol. 14(1), 1984, pp. 143-148.

[Lan90] Langerak, R. "View updates in Relational Databases with Independent Schema Interface", ACM Transactions on Database Systems Vol. 15(1), March 1990, pp.40-66.

[LLS93] Laurent, D.; Luong, V.P.; Spyratos, N. "Updating Intensional Predicates in Deductive Databases" Proc. 9th International Conference on Data Engineering (ICDE'93), Vienna, Austria, April 1993, pp. 14-21.

[LS91] Larson, J.; Sheth, A. "Updating Relational Views Using Knowledge at View Definition and View Update Time" Information Systems, Vol. 16(2), 1991, pp. 145-168.

[LT97] Lobo, J.; Trajcevski, G. "Minimal and consistent evolution in knowledge bases" Journal of Applied Non-Classical Logics, 7(1-2), 1997, pp. 117-146.

[LlT84] Lloyd, J. W.; Topor, R. W. "Making Prolog More Expressive", Journal of Logic Programming, 1984, Vol. 1(3), pp. 225-240.

[Maa98] Maabout, S. "Maintaining and Restoring Database Consistency with Update Rules" Workshop DYNAMICS'98 (post-conf. Work. JICSLP'98), Manchester, 1998, pp. 59-74.

[Mas84] Masunaga, Y. "A Relational Database View Update Translation Mechanism", Proc. 10th. International Conference on Very Large Data Bases (VLDB'84), Singapore, 1984, pp. 309-320.

[ML91] Moerkotte, G.; Lockemann, P.C. "Reactive Consistency Control in Deductive Databases" ACM Transactions on Database Systems, 16(4), 1991, pp.670-702

[MT99a] Mayol, E.; Teniente, E. "Addressing Efficiency Issues During the Process of Integrity Maintenance", 10th International Conference on Database and Expert Systems Applications (DEXA'99), Florence, Italy, September 1999, LNCS-1677, pp. 270-281.

[MT99b] Mayol, E.; Teniente, E. "A Survey of Current Methods for Integrity Constraint Maintenance and View Updating", Advances in Conceptual Modelling (LNCS-1727), Proc. of the First Workshop on Evolution and Change in Data Management (ECDM'99), Post-ER'99 Conference Workshops, Paris, November 1999, pp. 62-73.

[MT00] Mayol, E.; Teniente, E. "Dealing with Modification Requests during View Updating and Integrity Constraint Maintenance", Proc. of the International Symposium on Foundations of Information and Knowledge Systems (FoIKS'00), LNCS-1762, Burg (Spreewald), Germany, Feb. 2000, pp. 192-212.

[MW88] Manchanda, S.; Warren, D.S. "A Logic-based Language for Database Updates", In Foundations of Deductive Databases and Logic Programming (J. Minker Ed.), Morgan-Kaufmann Publications, 1988, pp. 363-394.

[Oli91] Olivé, A. "Integrity Checking in Deductive Databases", Proc. of the 17th VLDB Conference, Barcelona, Catalonia, 1991, pp. 513-523.

[PO95] Pastor, J.A.; Olivé. A. "Supporting Transaction Design in Conceptual Modeling of Information Systems", 7th Int. Conf. on Advanced Information Systems (CAiSE'95), Jyväskylä, Finland, June 1995, pp.40-53.

[Sag83] Sagiv, Y. "A Characterization of Globally Consistent Databases and Their Correct Access Path" ACM Transactions on Database Systems, Vol. 8(2), June 1983, pp. 266-286.

[Sch96] Schewe, K.D. "Tailoring Consistent Specializations as a Natural Approach to Consistency Enforcement", 6th Int. Workshop on Foundations of Models and Languages for Data and Objects: Integrity in Databases (FMLDO'96), Dagstuhl, Sept. 1996, pp. 73-85.

[Sch98] Schewe, K.D. "Consistency Enforcement in Entity-Relationship and Object-Oriented Models", Data & Knowledge Eng., Vol. 28(1), 1998, pp.121-140

[Sch00] Schewe, K.D. "Controlled Automation of Consistency Enforcement", 15th IEEE Int. Conf. On Automated Software Engineering (ASE'00), Grenoble, France, September 2000, pp.265-268.

[Sel95] Seljée, R. "A New Method for Integrity Constraint Checking in Deductive Databases" Data & Knowledge Engineering Vol. 15(1), 1995, pp.63-102.

[SM89] Subieta, K.; Missala, M. "View Updating Through Predefined Procedures" Information Systems, Vol. 14(4), 1989, pp. 291-305

[Spy80] Spyratos, N. "Translation Structures of Relational Views", Proc. 6th. Int Conf. On Very Large Databases (VLDB'80), Montreal, 1980, pp. 411-416.

[ST98] Schewe, K.D.; Thalheim, B. "Limitations of Rule Triggering Systems for Integrity Maintenance in the Context of Transition Specifications ", Acta Cybernetica, Vol. 13(3), 1998, pp. 277-304.

[ST99] Schewe, K.D.; Thalheim, B. "Towards a theory of consistency enforcement ", Acta Informatica, Vol. 36(2), 1999, pp. 97-141.

[Ten00] Teniente, E. "Deductive Databases", In Advanced Database Technology and Design (M.Piattini, O.Díaz Editors), Artech House Inc., 2000, pp. 91-136.

[TFC83] Tucherman, L.; Furtado, A.L.; Casanova, M.A. "A Pragmatic Approach to Structured Database Dessign", Proc. 9th Int Conf. On Very Large Databases (VLDB'83), Florence, 1983, pp. 219-231.

[TO95] Teniente, E.; Olivé, A. "Updating Knowledge Bases while Maintaining their Consistency", The VLDB Journal, Vol. 4(2), 1995, pp. 193-241.

[TU95] Teniente, E.; Urpí, T. "A Common Framework for Classifying and Specifying Deductive Database Updating Problems", 11th Int. Conf. on Data Engineering (ICDE), Taipei (Taiwan), 1995, pp. 173-183.

[Win90] Winslett, M. "Updating Logical Databases", Cambridge Tracts in Theoretical Computer Science Nº 9, 1990.

[Wüt93] Wüthrich, B. "On Updates and Inconsistency Repairing in Knowledge Bases", Int. Conference on Data Engineering (ICDE'93), Vienna 1993, pp.608-615.