Formulating View Update Translation as Constraint Satisfaction

Hua Shu
Institut für Informatik
Universität Hannover
Lange Laube 22, D-30159 Hannover, Germany
telephone: +49 511 762 4960
fax: +49 511 762 4961
hs@informatik.uni-hannover.de

Abstract

View update is the problem of translating update requests against a view into update requests against the base data. Previous approaches to view update translation have been mainly focused on studying the conditions under which an exact, unique translation of a view update request exists. The conditions are satisfied only by a small class of view updates. The approaches to enumerating possible translations are essentially procedural, in the sense that problem formalization and problem solving are not separated. Even though the role of database and application semantics is recognized, it is usually difficult to incrementally add semantic information to resolve ambiguity in view update translation. A slight change of a problem specification would render the algorithms inapplicable. In this paper, we present a declarative approach to the view update problem. Using *conditional tables* to represent views, we transform a view update problem into *constraint satisfaction problems* (CSPs). Solutions to the CSPs correspond to possible translations of the view update. Semantic information to resolve ambiguity can be added in terms of additional constraints into the CSPs. This approach enables us to make use of the rich results of the CSP research to analyze and solve view update problems.

Keywords: View update, constraint satisfaction, conditional tables, relational databases.

1 Introduction

The view update problem refers to the problem of determining the translation of an update request against a view into updates against the base data. Given a view update request, the corresponding updates of the base data may not be unique, not exist or be ill-defined. They may create inconsistencies in the base data and have side effects on the involved views. The essential task of view update problem has therefore been the determination of the *unique* translation of a view update request that both maintains consistency and is free of side effects. There are two frequently referred approaches to view updates. One is to define views as abstract data types so that a view definition includes all permissible view updates together with their translators [34]. A problem with this approach is that for each view, the view update operations have to be explicitly defined and the definitions have to be maintained during the lifetime of the database. Another approach is to define general translation procedures [13, 22, 28, 36]. The view-dependency approach [13], for instance, explores the structure of the dependencies of the underlying relation schema (rather than instances of the relational schemas) to determine unique translation of a view update request. One of the problems with this approach is that the approach translates view updates only into base data updates of the same type (e.g. deletions into deletions). Another limitation is its incompleteness. Since the syntactic conditions are rather stringent, many view updates do not satisfy the conditions. For a view update not satisfying the conditions, there may still be an exact and unique translation with respect to a specific state of the database. Yet the approach does not offer any way to find such a translation (see Section 7).

The general translation procedures proposed so far are mostly procedural and ad hoc, in the sense that problem formalization and problem solving are not separated. Thus a slight change of the problem specification (for instance to add semantic information to resolve ambiguity) would render the algorithms inapplicable. On the other hand, the view update problem can intuitively be understood as constraint satisfaction, where the constraints come from a number of sources, including view definitions, integrity constraints on the database, the authorization restriction, and the requirement of excluding side effects. Despite its popularity as a problem solving method, constraint satisfaction has not been used to formulate translation procedures of view updates (to our knowledge).

In this paper, we present a declarative approach to the view update problem by transforming a view update problem into constraint satisfaction problems (CSPs). Each solution to the CSPs corresponds to one possible translation of the view update. The transformation into CSPs makes it possible to enumerate the possible translations of a view update request based on the given semantic information. Often there are multiple solutions to the CSPs. Each solution corresponds one possible translation of the view update. Semantic information to resolve ambiguity can be added in terms of additional constraints into the CSPs. Some additional criteria may be used to choose the optimal translations. For instance, one may use a criterion of *minimal updates* to prefer a translation performing minimal changes to the base data. The proposed approach enables us to utilize the rich results of the growing CSP research, including the large number of constraint satisfaction algorithms and techniques [37].

In this work, views are restricted to contain one relation scheme. The view updates are single-tuple updates. View definitions are in terms of relational operators projections, selections and joins involving only equalities. A view insertion request may be translated into insertions or modifications. A view deletion request may be translated into deletions and modifications. A view modification request may be translated into insertions and modifications.

This paper is organized as follows. In Section 2, we describe the notions of conditional tables from [20], with three-valued extensions, and update transactions. Conditional tables are used to represent views. We present the basic ideas behind this approach in Section 3. In Section 4 a key problem of this approach is defined. In Section 5 the details of the approach are presented. Section 7 discusses some related work and compares them with this approach. Section 8 concludes the paper and discusses the relationship of the proposed approach to the previous ones.

2 Preliminaries

We assume that a countably infinite set of attributes is fixed. Each attribute A has a domain, denoted $Dom(A)$. There is a countably infinite set $N_1 = \{\delta_1, \delta_2, ...\}$ of *null values* used to represent partial information. They are interpreted as "value exists, but unknown". A special value η, called *non-existing value*, is used to represent the lack of values. It is assumed that for each attribute A, $Dom(A)$ does not contain nulls nor the non-existing value. We define $\mathbf{V}$ to be a countably infinite set of *variables*. Each variable x is associated with a domain $D(x)$ which may contain nulls as well as the non-existing value. The assignment of a null value to variable x means that the value of x exists, but is unknown. The assignment of the non-existing value to variable x means that the value of x simply does not exist.

For a finite set $\mathbf{V}$ of variables, a *valuation* over $\mathbf{V}$ is a mapping $\mathbf{v}$ from variables in $\mathbf{V}$ to values in the associated domains of the variables. Such a valuation is expressed in form of $\{x_1/a_1, x_2/a_2, ..., x_n/a_n\}$, where $x_1, ..., x_n$ is a listing of $\mathbf{V}$ and $a_i = \mathbf{v}(x_i) \in D(x_i)$ for each $i \in (1, n)$. A valuation $\mathbf{v}$ is said to be *partial* with respect to $\mathbf{V}$ if for some variable $x \in \mathbf{v}$, $\mathbf{v}(x) = \eta$.

2.1 Conditional Tables

As an extended relational model, *conditional table* was originally proposed in [20] to handle incomplete information in relational databases. A *table* over a finite set R of attributes is a finite set of free tuples over R such that each variable occurs at most once. In other words, a table is relation with constants and variables in which no variable occurs twice. For a tuple t and a valuation $\mathbf{v}$, $\mathbf{v}(t)$ denotes the resulting tuple after t being evaluated with respect to the valuation $\mathbf{v}$ of the variables. If a valuation $\mathbf{v}$ assigns the non-existing value to some variable in tuple t, then we say that $\mathbf{v}(t)$ is evaluated to be *non-existing*, denoted $\mathbf{v}(t) = \eta$.

A conditional table is a table whose tuples are associated with logical formulas. The logical formulas (called *conditions*) are expressions built up by means of Boolean connectives $\neg, \wedge, \vee$ and $\Rightarrow$ from constants **true**, **false**, and equality atoms of the forms $x = y$ and $x = c$, where x and y are variables and c is a constant.

Definition 1 *A* conditional table *(or* C-table *for short) is a tuple $T_c = (T, \phi)$, where*

- *T is a table,*
- *ϕ is a mapping over T that associates a condition $\phi(t)$ with each tuple t of T. $\phi(t)$ is called the local condition associated with t. A* c-tuple *is a tuple t of T together with local condition $\phi(t)$.*

The local conditions are represented under a column labeled *con*. Table 1 shows an example of C-tables. Note that the local conditions may contain variables not appearing in the table part of the C-tables. A C-table is said to be a *horn c-table* if the local conditions are conjunctions of atomic equalities.

We denote the evaluation of a condition C with respect to a valuation $\mathbf{v}$ by $V^{\mathbf{v}}(C)$. With respect to valuation $\mathbf{v}$, C can be satisfied (denoted $V^{\mathbf{v}}(C) = \mathbf{T}$), falsified (denoted $V^{\mathbf{v}}(C) = \mathbf{F}$) or *undefined* (denoted $V^{\mathbf{v}}(C) = \mathbf{U}$). The evaluation of a condition is defined recursively as follows:

Table 1: A conditional table (C-table)

A	C	con
t	c	$x = u$
s	c	$y = u$
t	c'	$x = v$
s	c'	$y = v$

1. For an equality atom $x = y$, $V^{\mathbf{v}}(x = y) = \mathbf{U}$ if $\mathbf{v}(x) = \eta$ or $\mathbf{v}(y) = \eta$. $V^{\mathbf{v}}(x = y) = \mathbf{T}$ if $\mathbf{v}(x) \neq \eta$, $\mathbf{v}(y) \neq \eta$ and $\mathbf{v}(x) = \mathbf{v}(y)$. Otherwise, $V^{\mathbf{v}}(x = y) = \mathbf{F}$.
2. **true** is always evaluated T and **false** always $\mathbf{F}$.
3. The evaluation of conditions with Boolean connectives is based on the *strong Kleene logic* [27]. That is, $V^{\mathbf{v}}(\neg p) = \neg V^{\mathbf{v}}(p)$, $V^{\mathbf{v}}(p \wedge q) = V^{\mathbf{v}}(p) \wedge V^{\mathbf{v}}(q)$, and $V^{\mathbf{v}}(p \vee q) = V^{\mathbf{v}}(p) \vee V^{\mathbf{v}}(q)$, where the connectives $\neg, \wedge$ and $\vee$ on the right-hand sides of the above equations are defined by the truth tables shown in Table 2 (The truth table for $\Rightarrow$ can be derived from those of $\neg$ and $\vee$).

We say that two conditions C_1 and C_2 are *equivalent* if and only if $V^{\mathbf{v}}(C_1) = V^{\mathbf{v}}(C_2)$ for any valuation $\mathbf{v}$. Conditions equivalent to **true** can be omitted in the c-tables. If all local conditions in a C-table are **true**, then the *con* column can be omitted.

Table 2: The truth tables of the connectives $\neg, \wedge$ and $\vee$

p	$\neg p$
T	**F**
F	**T**
U	**U**

$\wedge$	**T**	**F**	**U**
T	**T**	**F**	**U**
F	**F**	**F**	**F**
U	**U**	**F**	**U**

$\vee$	**T**	**F**	**U**
T	**T**	**T**	**T**
F	**T**	**F**	**U**
U	**T**	**U**	**U**

The result of evaluating C-table $T_c = (T, \phi)$ with respect to valuation $\mathbf{v}$, denoted $\mathbf{v}(T_c)$, is defined as follows:

$$\mathbf{v}(T_c) = \{\mathbf{v}(t) \mid \ t \in T,\ V^{\mathbf{v}}(\phi(t)) = \mathbf{T}, \text{ and } \mathbf{v}(t) \neq \eta\} \tag{1}$$

So $\mathbf{v}(T_c)$ contains all such $\mathbf{v}(t)$ that $\mathbf{v}$ satisfies the local condition $\phi(t)$ associated with $t \in T$ and $\mathbf{v}(t)$ is not evaluated to non-existing. We call $\mathbf{v}(T_c)$ the *state* of T_c with respect to $\mathbf{v}$. For a set I_c of C-tables, $\mathbf{v}(I_c) = \{\mathbf{v}(T_c) \mid T_c \in I_c\}$.

The collection of all possible states of T_c is defined as:

$$rep(T_c) = \{\mathbf{v}(T_c) \mid \text{valuation } \mathbf{v} \text{ satisfies } \Phi_T\}. \tag{2}$$

For two C-tables T_1 and T_2, they are said to be *rep-equivalent* if $rep(T_1) = rep(T_2)$, or $rep(T_1)$ and $rep(T_2)$ are identical after renaming the null values. Given a C-table T_c, if we (a) replace each of the conditions in T_c by an equivalent one, (b) delete all $t \in T_c$ such that $\phi(t)$ is equivalent to **false**, and (c) replace some $t_1, ..., t_k \in T_c$ such that $t_1[X] = ... = t_k[X]$, where X is the set of attributes in T_c, by a single tuple t such that $t[X] = t_1[X]$ and $\phi(t) = \vee_{i=1}^{k} \phi(t_i)$, then the resulting C-table will be rep-equivalent to T_c [20]. If the resulting C-table does not contain different c-tuples agreeing on all the attributes of T_c, then it is said to be *normalized*, denoted T_c^0 [20].

Suppose that C-tables $T_c = (T, \phi_1)$ and $W_c = (W, \phi_2)$ are given, which are defined on sets X and Z of attributes, respectively. The definitions of relational operators on C-tables are as follows:

- *Projection*: The projection of T_c on Y, denoted $\pi_Y(T_c)$, is the C-table $(\pi_Y(T), \phi_1')^0$ where ϕ_1' is a mapping over $\pi_Y(T)$ that for each tuple $t \in \pi_Y(T)$ such that $t = t'[Y]$ for some $t' \in T$, $\phi_1'(t) = \phi_1(t')$.
- *Selection*: Selection is performed by adding new conjuncts to the local conditions. For selection condition E, which is of the form $A = c$ or $A = B$ where A and B are attributes and c is a constant, the selection of T_c by condition E, denoted $\sigma_E(T_c)$, is defined as $(\sigma_E(T), \phi')^0$ where

$$\sigma_E(T) = \{\sigma_E(t) \mid t \in T\},$$

where $\sigma_E(t)$ is the c-tuple on X with $\sigma_E(t)[X] = t[X]$. For any $t \in \sigma_E(T)$,

$$\phi'(t) = \phi_1(t) \wedge E(t).$$

$E(t)$ is the result of substituting $t(A)$ for A in E, for every $A \in X$. In Table 3, the result of selection $\sigma_{C=2}R$ is illustrated.

- *Union*: Suppose that T_c and W_c use distinct sets of variables. If $X = Z$, then the union of T_c and W_c is defined as

$$T_c \cup W_c = (T \cup W, \phi)^0,$$

where for any $t \in T \cup W$,

$$\phi(t) = \begin{cases} \phi_1(t) & \text{if } t \in T - W, \\ \phi_2(t) & \text{if } t \in W - T, \\ \phi_1(t) \vee \phi_2(t) & \text{if } t \in T \cap W. \end{cases}$$

- *Join*: The natural join of two C-tables T_c and W_c, denoted $T_c \bowtie W_c$, is defined as $(T \bowtie W, \phi')^0$ where

$$T \bowtie W = \{t \bowtie w \mid t \in T \wedge w \in W\},$$

where $t \bowtie w$ is the tuple on $X \cup Z$ such that its projection on attribute A, denoted $(t \bowtie w)(A)$, is

$$(t \bowtie w)(A) = \begin{cases} t(A) & \text{if } A \in X, \\ w(A) & \text{if } A \in Z - X. \end{cases}$$

For any $t \bowtie w \in T \bowtie W$ where $t \in T$ and $w \in W$,

$$\phi'(t \bowtie w) = \phi_1(t) \wedge \phi_2(w) \wedge \bigwedge_{A \in X \cap Z} (t(A) = w(A)).$$

In Table 3, the join $R \bowtie R'$ is illustrated as an example.

Table 3: Relational operations on C-tables

R

A	B	C
0	1	x
y	z	1
2	0	v

R'

C
2

R ⋈ R'

A	B	C	con
0	1	x	$x = 2$
2	0	v	$v = 2$

$\sigma_{C=2}$R

A	B	C	con
0	1	x	$x = 2$
2	0	v	$v = 2$

The above definitions of the relational operations generalize straightforward to the cases with more than two C-tables. As shown in [18] (Theorem 5.1), the time taken to evaluate the operations on a set of C-tables is bounded by a polynomial of the maximal size of the C-tables.

Given a set I_c of C-tables and a query q, we denote the result of the query on the set I_c of C-tables by $q(I_c)$. It has been proven (as Theorem 7.1 in [20]) that for any PSUJ operator q and any set I_c of C-tables,

$$\mathbf{v}(q(I_c)) = q(\mathbf{v}(I_c)) \tag{3}$$

holds for any valuation $\mathbf{v}$ of the variables in I_c. That implies, *for any PSUJ query q and any set I_c of C-tables, $q(I_c)$ can be represented by a C-table* [20]. Generally speaking, all relational operators can be defined in such a way that (3) holds (Theorem 9.2 in [20]). Note that although null values and the non-existing value are not considered in [20], the above result still applies when they are taken into account. This is because the proof of the result has only to do with the definitions of the relational operators, which are not affected by the introduction of null values and the non-existing value.

2.2 Update transactions

For simplicity, we adopt update operations that deal with tuples in a single relation. The following notions are used to denote the primitive update operations:

- $ins(t, T)$: inserting tuple t into relation T.

- $del(t, T)$: deleting tuple t from T.
- $mod(t \to t', T)$: replacing tuple t by tuple t' in relation T.

The semantics of an update, described in terms of the *effect* of an update μ (denoted *eff*(μ)), is defined by a mapping from old instances to new instances of the relation, i.e.

- $\mathit{eff}(ins(t, T)) = T \cup \{t\}$.
- $\mathit{eff}(del(t, T)) = T - \{t\}$.
- $\mathit{eff}(mod(t \to t', T)) = (T - \{t\}) \cup \{t'\}$.

Thus the modification operation is redundant; it can be simulated by deletions and insertions.

An insert-delete-modify (IDM) transaction $\mathcal{T}$ is a finite sequence of updates. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are IDM transactions, then the concatenation $\mathcal{T}_1\mathcal{T}_2$ is also an IDM transaction. The effect of $\mathcal{T}_1\mathcal{T}_2$ is the composition of the effects of $\mathcal{T}_1$ and $\mathcal{T}_2$. Two transactions are *equivalent* if they have the same effects on every state of the database.

Let T_c be a C-table and T_1 and T_2 be two states of the C-table such that $T_1 = \mathbf{v}_1(T_c)$ and $T_2 = \mathbf{v}_2(T_c)$ for valuations $\mathbf{v}_1$ and $\mathbf{v}_2$. If T_2 is the effect of U with respect to T_1, we say that T_2 is *achievable* from T_1 by means of an IDM transaction U, denoted by $T_2 = U(T_1)$. Let $\mathbf{v}_1$ and $\mathbf{v}_2$ be two valuations. The number of different instantiations between $\mathbf{v}_1$ and $\mathbf{v}_2$ is called the *distance* between $\mathbf{v}_1$ and $\mathbf{v}_2$.

If $T_1 = \mathbf{v}_1(T_c)$ and $T_2 = \mathbf{v}_2(T_c)$ for some C-table T_c, and T_2 is achievable from T_1, then the distance between $\mathbf{v}_1$ and $\mathbf{v}_2$ can be used to measure the changes made to T_1 in order to achieve T_2. Suppose that $T_3 = \mathbf{v}_3(T_c)$ for some valuation $\mathbf{v}_3$ and T_3 is achievable from T_1 as well. If the distance between $\mathbf{v}_3$ and $\mathbf{v}_1$ is "shorter" than that between $\mathbf{v}_2$ and $\mathbf{v}_1$, then based on the criterion of minimal updates, $\mathbf{v}_3$ (and therefore T_3) will be prefered.

There is generally an equivalent class of IDM transactions by which one state of the c-table is achievable from the other (the class can also be empty). Here we only consider how one instance in the equivalent class can be constructed and ignore the optimization issue [1]. Let U be one instance in the equivalent class of IDM transactions by which T_2 is achievable from T_1. Then U can be constructed as follows. Initially, U is empty. For each tuple $t \in T_c$,

- if $\mathbf{v}_1(t) = \eta$ and $\mathbf{v}_2(t) \neq \eta$, then $ins(t, T_1)$ is appended to U.
- if $\mathbf{v}_2(t) = \eta$ and $\mathbf{v}_1(t) \neq \eta$, then $del(t, T_1)$ is appended to U.
- Otherwise, if $\mathbf{v}_1(t) \neq \eta, \mathbf{v}_2(t) \neq \eta$, and $\mathbf{v}_1(t) \neq \mathbf{v}_2(t)$, then $mod(\mathbf{v}_1(t) \to \mathbf{v}_2(t), T_1)$ is appended to U.

If $T_1 = \mathbf{v}_1(T_c)$ and $T_2 = \mathbf{v}_2(T_c)$ and T_2 is achievable from T_1 by means of an IDM transaction U constructed as specified above, then the distance between $\mathbf{v}_1$ and $\mathbf{v}_2$ determines the number of operations in U.

Given a relation T, one can use variables to replace certain values of the relation so that the possible states of the resulting table represent the effects of all deletions or modifications against T. For insertions against T, one has to insert a free tuple into T in order to represent the effects of the insertions against T. We illustrate those using an example.

Example 1 Consider instances of relations $T_1(A, B)$ and $T_2(B, C)$ shown in Table 4. The domains of the attributes are $Dom(A) = \{a, a'\}$, $Dom(B) = \{b, b'\}$ and $Dom(C) = \{c, c'\}$.

Table 4: The example relations

T_1

A	B
a	b
a'	b

T_2

B	C
b	c
b'	c'

To represent the effects of modification operations against the relations, we introduce variables to replace the values in the relations. The resulting tables are shown in Table 5. The valuation of the variables is

$$\mathbf{v}_1 = \{t/a, s/a', x/b, y/b, u/b, v/b', z/c, r/c'\}.$$

For the obtaining set $\{T_1^V, T_2^V\}$ of tables, $\mathbf{v}_1(T_1^V) = T_1$ and $\mathbf{v}_1(T_2^V) = T_2$. The domains of the variables are the domains of the corresponding attributes, i.e.

$$D(t) = D(s) = Dom(A), D(x) = D(y) = D(u) = D(v) = Dom(B), D(z) = D(r) = Dom(C). \quad (4)$$

Table 5: Introducing variables into base relations

T_1^V

A	B
t	x
s	y

T_2^V

B	C
u	z
v	r

Then *eff*$(mod(t \to t', T_1)) \in rep(T_1^V)$ and *eff*$(mod(t \to t', T_2)) \in rep(T_2^V)$ for any modification $mod(t \to t', T_1)$ against T_1 and $mod(t \to t', T_2)$ against T_2.

To represent the effects of all deletion operations against the relations, we make use of the tables T_1^V and $_2^V$, and extend the domain of each variable with the non-existing value, i.e.

$$D(t) = D(s) = Dom(A) \cup \{\eta\}, D(x) = D(y) = D(u) = D(v) = Dom(B) \cup \{\eta\}, D(z) = D(r) = Dom(C) \cup \{\eta\}. \quad (5)$$

To represent the effects of all insertion operations against the relations, we add to each of the relations a free tuple with only variables, as shown in Table 6. The domains of the variables are the domains of the corresponding attributes, i.e.

$$D(e) = Dom(A), D(f) = D(g) = Dom(B), D(h) = Dom(C).$$

Then *eff*$(ins(t, T_1)) \in rep(T_1')$ and *eff*$(ins(t, T_2)) \in rep(T_2')$ for any insertion $ins(t, T_1)$ against T_1 and $ins(t, T_2)$ against T_2.

Table 6: Adding free tuples to the relations

T_1'

A	B
a	b
a'	b
e	f

T_2'

B	C
b	c
b'	c'
g	h

□

Now we consider the representation of *relevant* updates against the base data to a view and define a few notions to be used in the following sections. A view may be defined using a *select-from-where* expression in the SQL syntax, as defined in [10], or equivalently, by an expression in the relational algebra. An attribute A is said to be *distinguished* in a view defined by a *select-from-where* expression if attribute A appears in the set of attributes specified by the *select* clause. An attribute A is said to be *exposed* in a view defined by a *select-from-where* expression if A is used in the selection condition specified by the *where* clause. The union of the distinguished and exposed attributes is called the *extended attribute set* of the view [19].

Definition 2 *Let I be a database and I_c be a set of tables such that $I = \mathbf{v}(I_c)$ for some valuation $\mathbf{v}$. We say that the variables in I_c are* sufficient *for view q defined over I if for any value a of an attribute in the extended attribute set of the view q, there is some variable x such that $x/a \in \mathbf{v}$. x is said to identify value a.*

A modification operation against the base relations of a view is relevant to the view (i.e. possibly affects the view) if the operation changes values of some attributes in the extended attribute set of the view. To capture the effects of modifications against the base relations, we only need to introduce variables to identify the values in the base relations so that the introduced variables are sufficient for the view.

Suppose that there is a view $\pi_A(T_1 \bowtie T_2)$ defined over the relations T_1 and T_2 in Example 1. To represent the effects of all modification operations against the base relations, we only need to introduce variables to replace the values of the attributes in the extended attribute set of the view, i.e. the attributes A and B, but not C.

Similar to the cases with modifications, deletions against the base relations are relevant to the view if the operations delete some tuple of such a base relation that at least one attribute of the relation is in the extended attribute set of the view. To capture the effects of a deletion against the base relations, we only need to introduce variables to identify the values in the base relations so that the introduced variables are sufficient for the view.

Definition 3 *Let I be a database and I_c be a set of tables such that $I = \mathbf{v}(I_c)$ for some valuation $\mathbf{v}$ and the variables in I_c are sufficient for view q defined over I. For each variable x identifying a value a of attribute A, if $D(x) = Dom(A)$, then I_c is called a* M-version *of I with respect to q. If $D(x) = Dom(A) \cup \{\eta\}$, then I_c is called a* DM-version *of I with respect to q.*

A DM-version (M-version) of I with respect to q can be used to represent the effects of deletions (resp. modifications) against I that are relevant to the view q.

An insertion operation against the base relations of a view is relevant to the view if the operation inserts some tuple into such a base relation that at least one attribute of the relation is in the extended attribute set of the view. To capture the effect of an insertion against the base relations, we need to introduce free tuples to identify the inserted tuples.

Definition 4 *Let I be a database and q be a view defined over I. We add a free tuple to each of the relations in I with either exposed or distinguished attributes of the view. Each of the free tuples is constructed as follows. For an exposed or distinguished attribute A, a variable x is introduced such that $D(x) = Dom(A) \cup \{\eta\}$. For any other attribute, a null value is introduced. The resulting set of tables is called an* I-version *of I with respect to q.*

An I-version of I with respect to q can be used to represent the effects of insertions against I that are relevant to the view q.

Definition 5 *Let I be a database and q be a view defined over I. Let I_c be a M-version of I with respect to q. We add a free tuple to each of the relations in I_c with either exposed or distinguished attributes of the view. Each of the free tuples is constructed as described in Definition 4. The resulting set of tables is called an* IM-version *of I with respect to q.*

An IM-version of I with respect to q can be used to represent the effects of insertions and modifications against I that are relevant to the view q.

2.3 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is represented as a triple $(\Gamma, D, \mathcal{C})$ where Γ is a finite set of variables, D is a function which maps each variable x in Γ to the domain of x, and $\mathcal{C}$ is the set of constraints on the variables in Γ. If each domain is finite, then the CSP is a finite constraint satisfaction problem. A *solution* to a constraint satisfaction problem $(\Gamma, D, \mathcal{C})$ is a valuation of the variables in Γ that satisfies all the members of $\mathcal{C}$. A CSP is *satisfiable* if a solution to the problem exists. An assignment of is said to be *complete* with respect to a CSP if the assignment assigns values to all the variables in the CSP. A solution to a CSP is not necessarily a complete assignment of the variables. A CSP may have multiple solutions. It may be required to find *all* solutions, or to find *optimal* solutions based on some domain-dependent criteria.

A *binary CSP* is a CSP with unary and binary constraints only (i.e. constraints involving one and two variables, respectively). A *general CSP* is a CSP with constraints not limited to unary and binary constraints. A general CSP can be transformed into a number of binary CSPs. The constraints considered in this paper are *conditions*. If a constraint C in $\mathcal{C}$ is a disjunctive condition $C_1 \vee ... \vee C_n$, then the CSP can be transformed into n sub-CSPs, each being the result of replacing C in the original CSP with one of the disjuncts. If a constraint C in $\mathcal{C}$ is a conjunctive condition $C_1 \wedge ... \wedge C_n$, then the connective $\wedge$ can be simply omitted. In this way, we can successively transform a general CSP into a number of binary CSPs. The solutions to each of the binary CSPs are also solutions to the original CSP. For the ease of presentation, we divide a set $\mathcal{C}$ of binary constraints into two disjoint sets $\mathcal{C}^+$ and $\mathcal{C}^-$. A binary CSP is then of the form $(\Gamma, D, \mathcal{C}^+, \mathcal{C}^-)$, where $\mathcal{C}^+$ contains equality atoms to be satisfied and $\mathcal{C}^-$ contains equality atoms not to be satisfied.

The method for solving CSPs is *search* i.e. to enumerate the assignments of the variables so as to find the solutions. A basic search algorithm is *simple backtracking*. The algorithm starts with a partial assignment to the variables (initially the assignment of one variable). The assignment is incrementally extended with the goal of satisfying as many constraints as possible. When some constraint cannot be satisfied with the current assignment, then the procedure backtracks and tries a different assignment for the latest considered variable. In the worst case, the simplest procedure of backtracking essentially searches through the space of all possible assignments to the variables. Let n denote the number of variables, e the number of constraints and a the largest domain size of the variables in a CSP. Then the time complexity of solving the CSP using the simple backtracking method is $O(a^n e)$. The space complexity is $O(na)$. There are a number of heuristic approaches to improving the performance of backtracking, for instance, by problem reduction, solution synthesis and problem decomposition. Due to space limit, we do not discuss the approaches in detail. Interested reader may consult the book by Tsang [37].

3 The Basic Ideas

Let I denote a database (state) on a database scheme with a set of constraints. It is assumed that I is consistent. Suppose that there is a view defined by some relational algebra expression q. The state of the view with respect to I is denoted by $q(I)$. We consider only the states without duplicates. A user may issue an update request μ against view $q(I)$; μ can be a deletion, an insertion or a modification. The updated view is denoted by $\mu(q(I))$. The problem considered here is how to translate μ into an IDM transaction U against the database I. In this section, we describe the basic ideas behind our approach to generating correct translations of view updates. First, we define what we mean by a translation of a view update request μ.

Definition 6 *An IDM transaction U against database I is said to be a* translation *of a view update request μ against view $q(I)$ if it satisfies the following criteria:*

1. *U preserves semantic consistency, i.e. the updated database $U(I)$ satisfies all the semantic constraints on the database scheme.*
2. *U satisfies the following condition with respect to μ: $q(U(I)) = \mu(q(I))$.*

This notion of translation is essentially the same as the notion of translation defined in [17]. The problem with side effects does not occur in updating views defined by only selection and projection operators, but may occur in updating views defined by *join* operators. For simplicity, we assume that there are no semantic constraints. We defer the discussion on the cases with semantic constraints to Section 6.1.

A possible user requirement is that the user should be able to cancel the effect of every update on a view. This requirement has been captured in [4] using a notion of acceptability. It can be formulated in our terminology as follows.

Definition 7 *A translation U of a update request μ against view $q(I)$ is said to be* acceptable *if $U(I) = I$ whenever $\mu(q(I)) = q(I)$.*

This notion of acceptability is called loop-preserving property in¡[17]. It is proved that for a restricted class of view updates (closed under inverse), there is unique, acceptable translation into updates of the base relations under maintenance of constant complement [4]. Acceptability is a weaker version of the notion consistency defined in [17].

The basic idea of our approach is to formulate the view update problem in terms of constraint satisfaction. Each solution to the CSPs corresponds to one possible translation of the view update request. The following example illustrates the translation of a deletion against a view involving joins.

Example 2 Consider the relation instances T_1 and T_2 shown in Table 4 and the view instance $\pi_{A,C}(T_1 \bowtie T_2)$ shown in Table 7.

Table 7: The view state

$V : \pi_{A,C}(T_1 \bowtie T_2)$

A	C
a	c
a'	c

The D-versions T_1^V and T_2^V of the relations are shown in Table 5, with the valuation $\mathbf{v}_1 = \{t/a, s/a', x/b, y/b, u/b, v/b', z/c, r/$
The view $\pi_{A,C}(T_1^V \bowtie T_2^V)$ is represented using the C-table shown in Table 8. The view state V shown in Table 7 is a state of the C-table with respect to $\mathbf{v}_1$.

Table 8: A C-table representing the view $\pi_{A,C}(T_1^V \bowtie T_2^V)$

$\pi_{A,C}(T_1^V \bowtie T_2^V)$

A	C	con
t	z	$x = u$
s	r	$y = u$
t	z	$x = v$
s	r	$y = v$

Consider the request $del((a,c), \pi_{A,C}(T_1 \bowtie T_2))$. The effect of this deletion is a relation containing only one tuple (a', c). How we want to find out the constraints under which the effect of this deletion is a state of the C-table $\pi_{A,C}(T_1^V \bowtie T_2^V)$. For simplicity, we keep the valuations of t, s, z and r as defined by $\mathbf{v}_1$ and consider only conditions involving variables x, y, u and v. Since only the evaluation of tuple (s, z) with respect to $\mathbf{v}_1$ results in tuple (a', c), the associated condition $y = u$ is the only one that should be satisfied; the other three conditions $x = u$, $x = v$ and $y = v$ should be falsified. We can construct the following binary CSP:

$$(\{x, y, u, v\}, D_\eta, \{y = u\}, \{x = u, x = v, y = v\}), \tag{6}$$

where D_η is defined in (5). There are a number of solutions to this CSP. Each of the solutions corresponds a possible translation of the view deletion.

Suppose that the criterion of minimal updates is adopted. The solutions with the shortest distance to the valuation $\mathbf{v}_1$ should be prefered. Only the following solution satisfies that condition:

$$\mathbf{v}_2 = \{x/\eta, y/b, u/b, v/b'\}. \tag{7}$$

The corresponding translation of the deletion request is the transaction containing only $del((a, b), T_1)$.

Semantic constraints can also be used to choose an appropriate solution. As an example, suppose that a deletion against the view should not be translated into modifications against relation T_1. This can be formulated in terms of two constraints: $x \neq \eta$ and $y \neq \eta$. Adding those constraints to the CSP (8), we obtain the following CSP:

$$(\{x, y, u, v\}, D_\eta, \{y = u, x \neq \eta, y \neq \eta\}, \{x = u, x = v, y = v\}), \tag{8}$$

There are two solutions to this CSP:

$$\mathbf{v}_3 = \{x/b, y/b', u/b', v/\eta\}, \mathbf{v}_4 = \{x/b', y/b, u/b, v/\eta\}.$$

According to the criterion of minimal updates, $\mathbf{v}_4$ is prefered. The corresponding translation is the transaction containing $mod((a, b) \to (a, b'), T_1)$ and $del((b', c'), T_2)$. □

The general procedure of our approach to solving the view update problem can be defined as follows.

Definition 8 *Given a database I, a view definition q and a primitive view update request μ,*

1. *Create tables that represent the search space of the possible translations of the view update request.*
   - *If the update request is a deletion, then let I_c be a DM-version of I with respect to q.*
   - *If the update request is an insertion, and it is intended to be translated into insertions against the base relations, then let I_c be an I-version of I with respect to q; if the insertion request is intended to be translated into modifications against the base relations, then let I_c be a M-version of I with respect to q.*
   - *If the update request is a modification, then let I_c be an IM-version of I with respect to q.*

   Let the valuation of the variables introduced in I_c w.r.t. I be $\mathbf{v}$. That is,

   $$\mathbf{v}(I_c) = I. \tag{9}$$

   We call the tuple (I, q, μ, I_c) a view update specification.
2. *Compute the view on I_c and represent it using a C-table T_c, i.e. $T_c = q(I_c)$. Based on the closure property (3) of C-tables, we can infer*

   $$\mathbf{v}(T_c) = \mathbf{v}(q(I_c)) = q(\mathbf{v}(I_c)) = q(I), \tag{10}$$

   which means that the state of $q(I_c)$ with respect to $\mathbf{v}$ is exactly the view state $q(I)$. Suppose that the effect of the view update is V', i.e.

   $$V' = \mu(q(I)). \tag{11}$$
3. *Construct CSPs based on V' so that for any valuation $\mathbf{v}$ of the variables in T_c,*

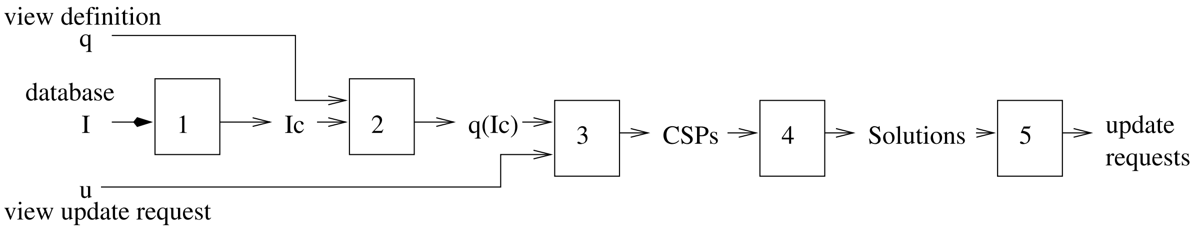
   $$V' = \mathbf{v}'(T_c) \tag{12}$$

   if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs.
4. *Solve the constructed CSPs.*
5. *Generate the translations of the view update based on the solutions to the CSPs. Thus for any solution $\mathbf{v}'$ such that*

   $$V' = \mathbf{v}'(T_c), \tag{13}$$

   a translation is such an IDM transaction U by which $\mathbf{v}'(I_c)$ is achievable from $\mathbf{v}(I_c)$ (i.e. I) (see Section 2.2 for the construction of such an IDM transaction), i.e.

   $$U(I) = \mathbf{v}'(I_c). \tag{14}$$



1. Introducing variables into the base relations
2. Computing the view and representing the result using a C-table
3. Generating CSPs based on the view and the view update request
4. Solving the CSPs
5. Generating the update requests based on the solutions

Figure 1: The procedure of solving the view update problem

The above procedure can be extended naturally to all update requests. Basically, the concatenation of a translation of an update μ_1 and a translation of an update μ_2 is a translation of $\mu_1\mu_2$. Figure 1 illustrates this procedure of solving the view update problem.

The construction of CSPs based on a given state of a C-table in the third step is a key technical problem for our approach. We shall show in Section 4 how this can be done. In the following, we prove that the translation procedure presented above generates all translations of a given view update request.

Theorem 1 *Let (I, q, μ, I_c) be a view update specification. The translation procedure described above generates all translations of μ.*

Proof:

First, we prove that what the procedure generates are indeed translations of μ. Let U be a generated IDM transaction based on the procedure. Now we prove that U has no side effects with respect to μ. Based on (14), $q(U(I)) = q(\mathbf{v}'(I_c))$. According to (3), $q(\mathbf{v}'(I_c)) = \mathbf{v}'(q(I_c))$. Note that $q(I_c) = T_c$. Based on (12), $\mathbf{v}'(q(I_c)) = \mathbf{v}'(T_c) = V'$. Based on (11), we have $q(U(I)) = \mu(q(I))$. So the transaction U has no side effects with respect to μ. According to Definition 6, we conclude that U is a translation of μ.

Secondly, we prove the completeness of the procedure, namely that for any translation, there is a corresponding solution to one of the constructed CSPs. Let U be a translation of μ. Then U has no side effect, i.e. $q(U(I)) = \mu(q(I))$. So $V' = \mu(q(I)) = q(U(I)) = q(\mathbf{v}'(I_c)) = \mathbf{v}'(q(I_c)) = \mathbf{v}'(T_c)$. According to the feature of the constructed CSPs, $\mathbf{v}'$ must be a solution to one of the constructed CSPs.

□

The following theorem states the condition under which the translations are acceptable.

Theorem 2 *Let (I, q, μ, I_c) be a view update specification. The translations of μ generated based on the procedure are acceptable if for any valuations $\mathbf{v}$ and $\mathbf{v}'$, $\mathbf{v}(q(I_c)) = \mathbf{v}'(q(I_c))$ iff $\mathbf{v} = \mathbf{v}'$.*

Proof:

Assume that $\mu(q(I)) = q(I)$. Let U be a translation of μ based on the procedure. We can easily prove that $U(I) = I$. Based on (11) and (12), we can infer $\mu(q(I)) = \mathbf{v}'(T_c)$. Based on this and (10), we infer $\mathbf{v}(T_c) = \mathbf{v}'(T_c)$. According to the given condition, $\mathbf{v} = \mathbf{v}'$. Thus $U(I) = \mathbf{v}'(I_c) = \mathbf{v}(I_c) = I$.

□

4 Constructing CSPs

Given a C-table T_c, we show in this section how to construct one or a few CSPs based on a state V of the C-table so that for any valuation $\mathbf{v}$ of the variables in T_c, $V = \mathbf{v}(T_c)$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs. First of all, let us illustrate the ideas using an example.

Example 3 Consider the C-table T_c and a state V of the C-table shown in Table 9. The domains of the variables are defined as follows:

$$D(t) = D(s) = \{a, a'\}, D(x) = D(y) = D(u) = D(v) = \{b, b'\}. \quad (15)$$

Table 9: A C-table T_c and a state V of the C-table

T_c

A	C	con
t	c	$x = u$
s	c	$y = u$
t	c'	$x = v$
s	c'	$y = v$

V

A	C
a	c
a'	c

Now we want to find out the necessary and sufficient conditions for any valuation $\mathbf{v}$ to satisfy $\mathbf{v}(T_c) = V$, i.e. to satisfy

$$\mathbf{v}(T_c) = V = \{(a, c), (a', c)\}. \tag{16}$$

The table part of the C-table T_c contains tuples $(t, c), (s, c), (t, c')$ and (s, c'), where t and s are variables. The above equation implies that the following should hold:

$$\{(a, c), (a', c)\} \subseteq \{\mathbf{v}((t, c)), \mathbf{v}((s, c)), \mathbf{v}((t, c')), \mathbf{v}((s, c'))\}.$$

This implies that the evaluation of variables t and s by $\mathbf{v}$ is such that either

$$t = a \text{ and } s = a',$$

or

$$t = a' \text{ and } s = a.$$

In either of these cases, the local conditions $x = u$ and $y = u$ associated with (t, c) and (s, c), respectively, have to be satisfied by $\mathbf{v}$, while the local conditions associated with (t, c') and (s, c') should not be satisfied. Under those conditions, (16) guarantees to hold. In other words, $\mathbf{v}$ should be a solution to one of the following binary constraint satisfaction problems:

$$(\{t, s, x, y, u, v\}, D, \{t = a, s = a', x = u, y = u\}, \{x = v, y = v\}),$$
$$(\{t, s, x, y, u, v\}, D, \{t = a', s = a, x = u, y = u\}, \{x = v, y = v\}),$$

where D is defined in (15). It is obvious that $\mathbf{v} = \{t/a, s/a', x/b, y/b, u/b, v/b'\}$. is a solution to the first CSP (17) and it satisfies the condition $\mathbf{v}(T_c) = V$. □

The following definition describes how in general CSPs can be constructed based on the state of a given C-table.

Definition 9 *Let V be a state of a C-table $T_c = (T, \phi)$ where table T is defined on a set X of attributes and $V = \mathbf{v}(T_c)$ for some valuation $\mathbf{v}$. A constraint satisfaction problem $(\Gamma, D, \mathcal{C})$ based on view state V can be constructed in the following way:*

- *Γ is the set of variables in T_c,*
- *D is the function defining the domain of the variables in T_c,*
- *If V is non-empty, and there is some partial valuation $\mathbf{v}_p$ that assigns values only to the variables in T such that*

$$V \subseteq \mathbf{v}_p(T) \in rep(T), \tag{17}$$

then $\mathcal{C}$ is the smallest set satisfying the following conditions:

1. *Let us define a subset of table T as follows:*

$$S(T, V, \mathbf{v}_p) = \{t \in T \mid \exists t' \in V \text{ so that } t' = \mathbf{v}_p(t)\}. \tag{18}$$

For each tuple $t \in V$, let $t_1, ..., t_k \in S(T, V, \mathbf{v}_p)$ be all the tuples such that $\mathbf{v}_p(t_1) = ... = \mathbf{v}_p(t_k) = t$. Then the following constraint is in $\mathcal{C}$:

$$\vee_{i=1}^{k}(\phi(t_i) \wedge (\wedge_{A \in X} t_i(A) = t(A))).$$

2. *For each $t \in T - S(T, V, \mathbf{v}_p)$, $\neg\phi(t)$ is in $\mathcal{C}$.*

If V is empty, then $\mathcal{C}$ is the smallest set so that for each $t \in T$, $\neg\phi(t) \in \mathcal{C}$.

There may be multiple CSPs constructed based on the same state of a C-table, due to the different valuations $\mathbf{v}_p$ that satisfies condition (17). If the given C-table $T_c = (T, \phi)$ has only conditions that are equivalent to **true** and T contains no variables, then the constructed CSP is trivial; all constraints are equivalent to **true**.

The time complexity of constructing CSPs is determined by the size of the search space of the partial evaluations $\mathbf{v}_p$. Let n_T denote the number of variables in table T and a be the maximal size of the domains of the variables. Then the size of the search space is a^{n_T}. Thus the time complexity of constructing CSPs is $O(a^{n_T})$.

Now we prove that the CSPs constructed based on Definition 9 have the required property: for any valuation $\mathbf{v}$, $V = \mathbf{v}(T_c)$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs.

Theorem 3 *Let V be a state of a C-table $T_c = (T, \phi)$, where T is defined on a set X of attributes. For any valuation $\mathbf{v}$ of the variables in T_c, $\mathbf{v}(T_c) = V$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs based on Definition 9.*

Proof:

($\Rightarrow$) The case where $V = \emptyset$ is trivial. For the case where V is not empty, it is sufficient to show that for any $\mathbf{v}$ such that $\mathbf{v}(T_c) = V$, $\mathbf{v}$ satisfies all the constraints in one of the constructed CSPs. Consider a CSP $(\Gamma, D, \mathcal{C})$ constructed based on a valuation $\mathbf{v}_p$ that assigns values only to the variables in T and satisfies $V \subseteq \mathbf{v}_p(T) \in rep(T)$, as well as the condition:

$$\mathbf{v}_p \subseteq \mathbf{v}.$$

The above condition implies that

$$\mathbf{v}_p(t) = \mathbf{v}(t) \text{ for each } t \in T. \tag{19}$$

We verify that $\mathbf{v}$ satisfies all of the constraints in the CSP.

1. For each $t \in V$ and all $t_1, ..., t_k \in S(T, V, \mathbf{v}_p)$ such that $\mathbf{v}_p(t_1) = ... = \mathbf{v}_p(t_k) = t$, by definition, the following constraint is included in $\mathcal{C}$:
$$\vee_{i=1}^{k}(\phi(t_i) \wedge (\wedge_{A \in X} t_i(A) = t(A))). \tag{20}$$
Since $\mathbf{v}_p(t_1) = ... = \mathbf{v}_p(t_k) = t \in V = v(T_c)$, according to the evaluation of a C-table defined in (1), we infer that there must be some t_i $(1 \leq i \leq k)$ such that $\phi(t_i)$ is satisfied by $\mathbf{v}$ and $\mathbf{v}(t_i) = t$. $\mathbf{v}(t_i) = t$ implies that $\wedge_{A \in X} t_i(A) = t(A)$ is satisfied by $\mathbf{v}$. That is to say, at least one disjunct of the constraint (20) is satisfied by $\mathbf{v}$. So $\mathbf{v}$ satisfies the constraint (20).

2. For each $t \in T - S(T, V, \mathbf{v}_p)$, $\neg\phi(t)$ is included in $\mathcal{C}$ by definition. Note that $\mathbf{v}_p(t) \notin V$. Since (19) holds, $\mathbf{v}(t) \notin V$. So $\mathbf{v}$ cannot possibly satisfy $\phi(t)$; if so, then $\mathbf{v}(t) \in V$ would hold.

Thus $\mathbf{v}$ indeed satisfies all the constraints in the CSP. This concludes the proof of the left-to-right part of the theorem.

($\Rightarrow$) The case where $V = \emptyset$ is trivial. For $V \neq \emptyset$, suppose that $\mathbf{v}$ is a solution to one CSP constructed based on the C-table $T_c = (T, \Phi_T, \phi)$. We have to prove that $V = \mathbf{v}(T_c)$. According to the construction of the CSPs, there must be a partial valuation $\mathbf{v}_p$ that assigns values only to the variables in T such that $V \subseteq \mathbf{v}_p(T) \in rep(T)$ and the CSP is constructed based on $\mathbf{v}_p, V$ and T_c. For each tuple $t \in V$, since $\mathbf{v}$ satisfies the constraint $\vee_{i=1}^{k}(\phi(t_i) \wedge (\wedge_{A \in X} t_i(A) = t(A)))$, where $t_1, ..., t_k \in S(T, V, \mathbf{v}_p)$ are all the tuples such that $\mathbf{v}_p(t_1) = ... = \mathbf{v}_p(t_k) = t$, it must be that there is at least one tuple t_i $(1 \leq i \leq k)$ such that $\phi(t_i)$ is satisfied by $\mathbf{v}$ and $\mathbf{v}(t_i) = t \in V$. On the other hand, for any $t \in T - S(T, V, \mathbf{v}_p)$, $\neg\phi(t)$ is in $\mathcal{C}$. Thus $\mathbf{v}$ does not satisfy $\phi(t)$. Consequently $\mathbf{v}(t)$ cannot be in $\mathbf{v}(T_c)$. That is, $V = \mathbf{v}(T_c)$. This concludes the proof of the right-to-left part of the theorem.

□

5 Searching for View Update Translations

In this section, we illustrate our approach by some simple examples. Due to limited space, we consider the translations of deletion and insertion requests only.

5.1 Translating deletion requests

Recall Example 2. To handle the translation of deletion requests against the view V, we need to generate a DM-version of (T_1, T_2) by introducing variables to identify the attributes A, B and C in both T_1 and T_2. The results are tables T_1^V and T_2^V shown in Table 5. The domains of the variables are the domains of the identified attributes with the addition of the non-existing value, as defined in (5). The view can be represented using C-table

$\pi_{A,C}(T_1^V \bowtie T_2^V)$, shown in Table 10. Based on the valuation $\mathbf{v}_1 = \{t/a, s/a', x/b, y/b, u/b, v/b', z/c, r/c'\}$, we have $\mathbf{v}_1(T_1^V) = T_1, \mathbf{v}_1(T_2^V) = T_2$ and moreover,

$$\mathbf{v}_1(\pi_{A,C}(T_1^V \bowtie T_2^V)) = V. \tag{21}$$

Table 10: Recomputing the view after variables being introduced into the base relations

$\pi_{A,C}(T_1^V \bowtie T_2^V)$

A	C	con
t	z	$x = u$
s	z	$y = u$
t	r	$x = v$
s	r	$y = v$

Consider the request $del((a, c), \pi_{A,C}(T_1 \bowtie T_2))$. The effect of this deletion is a relation containing only one tuple (a', c). According to the construction defined in Definition 9, we find all partial valuations $\mathbf{v}_p$ assigning values to variables involved in the table part of C-table $\pi_{A,C}(T_1^V \bowtie T_2^V)$ (shown in Table 10) that satisfy the condition

$$\{(a', c)\} \subseteq \mathbf{v}_p(\{(t, z), (s, z), (t, r), (s, r)\}).$$

Such valuations are many. We consider only one of them: $\{t/a, s/a', z/c, r/c'\}$. Correspondingly, we can construct the CSP (8). Recall the solution $\mathbf{v}_2$ given in (7). The instantiations of T_1^V and T_2^V with respect to $\mathbf{v}_2$ are:

$$\mathbf{v}_2(T_1^V) = \{(a', b)\}, \mathbf{v}_2(T_2^V) = \{(b, c), (b', c')\}.$$

$\mathbf{v}_2(T_1^V)$ is the effect of $del((a, b), T_1)$. $\mathbf{v}_2(T_2^V)$ is the same as T_2. Thus a translation of the deletion request is the transaction containing only $del((a, b), T_1)$.

Note that the transformation into satisfiable CSPs is possible only when the deleted tuple in the view is the result of evaluating some tuples in the C-table $\pi_{A,C}(T_1^V \bowtie T_2^V)$ whose associated local conditions not equivalent to **true**.

Example 4 Consider the tables in Table 11. Not all the values of the exposed attributes in T_1 and T_2 are identified with variables in $T_1^{V'}$ and $T_2^{V'}$ (where y and v are variables). Consider the request $del((a, c), \pi_{A,C}(T_1 \bowtie T_2))$. The corresponding c-tuple to (a, c) in $\pi_{A,C}(T_1^{V'} \bowtie T_2^{V'})$ has no associated local condition. In this case, we has no way to transform the view update problem into a satisfiable CSP. □

Table 11: Introducing variables into base relations

T_1

A	B
a	b
a'	b

T_2

B	C
b	c
b'	c'

$\pi_{A,C}(T_1 \bowtie T_2)$

A	C
a	c
a'	c

$T_1^{V'}$

A	B
a	b
a'	y

$T_2^{V'}$

B	C
b	c
v	c'

$\pi_{A,C}(T_1^{V'} \bowtie T_2^{V'})$

A	C	con
a	c	
a'	c	$y = b$
a	c'	$b = v$
a'	c'	$y = v$

5.2 Translating insertion requests

There are two alternative ways to translate an insertion view request, dependent of the user-specified semantics. One is to translate it into modifications. The other is to translate it into insertions. If the insertion request is intended to be translated into modifications against the base relations, then we need to generate a M-version of the base relations. If the insertion request is intended to be translated into insertions into the base relations, then we need to generate an I-version of the base relations.

Example 5 Consider the relation instances shown in Table 4 and the corresponding view instance shown in Table 7. Suppose a view request is $ins((a', c'), \pi_{A,C}(T_1 \bowtie T_2))$. The effect of the update request is $\{(a, c), (a', c), (a', c')\}$. If the insertion request is intended to be translated into modification operations against the base relations, then we need to generate a M-version of T_1, T_2) by introducing variables to replace the values of attributes in the extended attribute set. The results are tables T_1^V and T_2^V shown in Table 5. The domains of the variables are defined in (4). The C-table representing the view is $\pi_{A,C}(T_1^V \bowtie T_2^V)$ shown in Table 10.

According to Definition 9, we can find a number of partial valuations of the variables involved in the table part of C-table $\pi_{A,C}(T_1^V \bowtie T_2^V)$ that satisfies the condition (17). Let us consider the valuation $\{t/a, s/a', z/c, r/c'\}$, which lead to the following CSP:

$$(\{t, s, x, y, z, u, v, r\}, D, \{t = a, s = a', z = c, r = c', x = u, y = u, y = v\}, \{x = v\}).$$

where D is defined in (4). There are two solutions to the CSP. One is $\{t/a, s/a', z/c, r/c', u/b, v/b, x/b, y/b\}$. The corresponding translation is $\{mod((b', c') \to (b, c'), T_2)\}$. The other solution is $\{t/a, s/a', z/c, r/c', u/b', v/b', x/b', y/b'\}$. The corresponding translation is $\{mod((a, b) \to (a, b'), T_1), mod((a', b) \to (a', b'), T_1), mod((b, c) \to (b', c), T_2)\}$. □

To translate an insertion request into insertions against the base relations, we have to construct the I-versions of the base relations according to Definition 4.

Example 6 Consider again the insertion request discussed in Example 5. Now it is intended to translate it into insertion operations against the base relations. In order to obtain (a', c') in the view, suppose that we add a tuple (a', f) in T_1 and tuple (w, c') in T_2, where f and w are new variables and the domains of these variables are:

$$D(f) = D(w) = Dom(B).$$

The essential task of the translation is to find out the values for variables f and w. Table 12 shows the tables with the free tuples inserted and the C-table representing the view.

Table 12: Example of view insertion

T_1^V

A	B
a	b
a'	b
a'	f

T_2^V

B	C
b	c
b'	c'
w	c'

$\pi_{A,C}(T_1^V \bowtie T_2^V)$

A	C	con
a	c	
a'	c	
a'	c'	$(b = w) \vee (b' = f) \vee (w = f)$

Based on Definition 9, we can construct the following CSP:

$$(\{f, w\}, D, \{(b = w) \vee (b' = f) \vee (w = f)\}, \{\}).$$

This CSP can be transformed into three binary CSPs. The solutions to the binary CSPs are:

$$\mathbf{v}_1 = \{w/b\}, \mathbf{v}_2 = \{f/b'\}, \mathbf{v}_3 = \{w/b, f/b\}, \mathbf{v}_4 = \{w/b', f/b'\}. \quad (22)$$

Based on the set-based semantics, the translated operations on the base relations corresponding to $\mathbf{v}_3$ is the same as that corresponding to $\mathbf{v}_1$, and the translated operations on the base relations corresponding to $\mathbf{v}_4$ is the same as that corresponding to $\mathbf{v}_2$. $\mathbf{v}_1$ gives no value for f. It can be assumed that f is assigned the non-existing value. Comparing $\mathbf{v}_1(T_2^V)$ with T_2, we can easily infer the translated operations on the base relations corresponding to $\mathbf{v}_1$ is $ins((b, c'), T_2)$. Similarly, we can infer that the translated operations on the base relations corresponding to $\mathbf{v}_2$ is $ins((a', b'), T_1)$.

□

Remark:

- It may not be possible to construct CSP according to Definition 9 at all. For instance, Consider the user insertion request $ins((a, c''), \pi_{A,C}(T_1 \bowtie T_2))$ where $\pi_{A,C}(T_1 \bowtie T_2)$ is shown in Table 7. It is not possible to construct any CSP according to Definition 9, since there is no way to find a valuation $\mathbf{v}_p$ that satisfies condition (17).
- If the user-specified semantics does not indicate the intended type of operations, then as a default, one can first consider the translation into insertions, since this is normally the case with a smaller search space. When it is not possible, we can consider the translation into modifications with a larger search space.

Example 7 Let us extend the domain of attribute A to include a new value a''. Consider the relation T_4 and a view V_4 defined on T_4 shown in Table 13. The resulting tables T_{4a}^V and $\pi_A(\sigma_{B=b}T_{4a}^V)$ after variables are introduced are shown in Table 13 as well. Consider the update request $ins((a''), V_4)$. The effect of this request is $\{(a), (a'')\}$. We cannot construct any CSP based on $\pi_A(\sigma_{B=b}T_{4a}^V)$ according to Definition 9, since it is impossible to assign both a and a'' to u. So we have to try the second option, namely to add a free tuple to T_{4a}^V and to re-compute the C-table representing the view. Let the free tuple be (v, y, δ_1), where v and y are new variables and δ_1 is a null value. The null value is used because the value of attribute C is irrelevant to the view. The resulting table after this free tuple being added to T_{4a}^V is T_{4b}^V. The C-table $\pi_A(\sigma_{B=b}T_{4b}^V)$ representing the view is also shown in Table 13. Now we can construct a number of CSPs based on $\pi_A(\sigma_{B=b}T_{4b}^V)$ according to Definition 9. The following CSP has a solution that yields the smallest number of updates:

$$(\{u, v, x, y\}, D, \{u = a, v = a'', x = b, y = b\}, \{\}).$$

where D is defined as follows:

$$D(u) = D(v) = D(w) = Dom(A) = \{a, a', a''\}, D(x) = D(y) = D(z) = Dom(B) = \{b, b'\}. \tag{23}$$

According to the solution $\{u/a, v/a'', x/b, y/b\}$ to the CSP, we derive that a translation of the view insertion request is $ins((a'', y, \eta), T_4)$.

Table 13: Inserting a free tuple with a null value and recomputing the C-table representing the view

T_4

A	B	C
a	b	c

$V_4 : \pi_A(\sigma_{B=b}T_4)$

A
a

T_{4a}^V

A	B	C
u	x	c

$\pi_A(\sigma_{B=b}T_{4a}^V)$

A	con
u	$x = b$

T_{4b}^V

A	B	C
u	x	c
v	y	δ_1

$\pi_A(\sigma_{B=b}T_{4b}^V)$

A	con
u	$x = b$
v	$y = b$

□

6 Discussion

6.1 Data dependencies

The model of conditional table proposed in [20] was extended with *global condition* by Grahne in order to represent the enforcement of data dependencies [20]. When the database with C-tables include dependencies, the additional information provided by the dependencies can be encoded in C-tables as well. Given a C-table T_c, the *completion* of a set F of full dependencies on the C-table, denoted $comp_F(T_c)$, can be represented by another C-table, using an extended *chase* procedure [18]. We omit the presentation of the detail of the procedure. It is worth mentioning that the completion of dependencies on a C-table introduces only changes to the global condition of the C-table. For example, consider a functional dependency $AB \to C$ over relation R in Table 3. The completion $comp_{AB\to C}(R)$ is shown in Table 14, with a new global condition.

For a set I_c of C-tables, the completion of a set F of full dependencies on I_c, denoted $comp_F(I_c)$, is defined as $comp_F(I_c) = \{comp_F(T_c) \mid T_c \in I_c\}$.

Our approach can be extended in a straightforward way to handle semantic constraints. When semantic constraints are specified, we only need to compute the completion of the constraints on the table I_c obtained in the first step of Definition 8. The completion can be represented using a set of C-tables. Let the set of C-tables be denoted by I_c'. Then the *view update specification* is tuple (I, q, μ, I_c'). We assume that the semantic constraints considered are restricted to functional dependencies, though the approach actually applies to full dependencies i.e. dependencies without existential quantifiers.

Consider Example 2. Suppose that a functional dependency $B \to C$ is associated with relation T_2. The completion of the dependency on T_2^V is to add the global condition $(u = v) \Rightarrow (z = r)$ (where $\Rightarrow$ is the logical

Table 14: The completion of functional dependency $AB \rightarrow C$ on table R

$comp_{AB \rightarrow C} R$

$((y = 0 \wedge z = 1) \Rightarrow x = 1) \wedge$
$((y = 2 \wedge z = 0) \Rightarrow v = 1)$

A	B	C
0	1	x
y	z	1
2	0	v

implication). The CSP constructed for the deletion is:

$$(\{x, y, u, v, z, r\}, D_\eta, \{y = u, (u = v) \Rightarrow (z = r)\}, \{x = u, x = v, y = v\}). \tag{24}$$

6.2 Complexity

The time/space complexity of computing a view on a C-table is polynomial of the size of the base relations. Let n_T denote the number of variables in table T and a be the maximal size of the domains of the variables. Then the time complexity of constructing CSPs is $O(a^{n_T})$. Let the number of constraints in the constructed CSPs be e. Then the time complexity of solving the CSPs is $O(a^{n_T} e)$. Thus this approach has a $O(a^{n_T} e)$ bound.

The computational cost of this approach has a lot to do with the number of variables. There needs a trade-off between the computational cost and the number of possible translations of the view update requests. When variables are sufficiently introduced, all possible translations of a view update can be found at a likely high computational cost. When fewer variables are introduced, only some possible translations of a view update can be found, but at a lower computational cost. This approach is especially suitable for those applications where a user is authorized to update certain part of the base data through a specific view. Taking into account the authorization may help reducing the number of introduced variables.

7 Related Work

View update in relational databases is considered to be very complicated. [13] provded a theoretical fundation characterizing the problems involved in view updating. The role of database semantics in solving the view update problem was recognized. Early approaches assume constraints on allowed data instances in form of functional dependencies [13, 9, 26]. Updates that violate these constraints should be rejected, thus to obtain the desired situation where exactly one method of updating stored data is possible.

In [16], previous work on view updates was surveyed and classified along two approaches. One is based on treating views as abstract data types so that a view definition includes all permissible view updates together with their translators. [5, 38]. The second approach is to define the so-called "general view update translator" based on either an analysis of the conceptual schema dependencies [13] or on the concept of view complement [4, 12, 17].

[13] uses two graphs to describe functional dependencies, keys, and subset constraints. It provides the syntactic conditions for exact and unique translations of view updates which can, in most case, be checked at view definition time without actually accessing the database. The procedures developed for translating view updates are limited in the sense that they translate view updates only into base data updates of the same type (e.g. deletions into deletions). Another limitation is that the approach is incomplete; it does not find all the exact and unique translations.

Table 15: View $\pi_{A,C}(T_1 \bowtie T_2)$ defined on relations T_1 and T_2

T_1

A	B
a	b
a'	b

T_2

B	C
b	c
b'	c'

$V : \pi_{A,C}(T_1 \bowtie T_2)$

A	C
a	c
a'	c

As an example, let's consider the view V, defined by $\pi_{A,C}(T_1 \bowtie T_2)$ over relations T_1 and T_2 shown in Table 15. Suppose that a user issues a request to delete tuple (a, c) from V. The question is how to translate this deletion request into update requests against the base relations T_1 and T_2. To determine a translation of the deletion request,

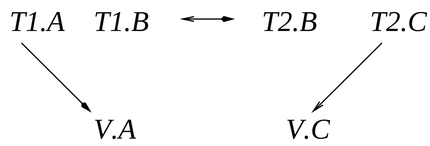


Figure 2: The view-dependency graph of the view $\pi_{A,C}(T_1 \bowtie T_2)$ defined on relations T_1 and T_2

it is important to identify the "clean source" of the tuple (a, c) [13]. The tuples (a, b) in T_1 and (b, c) in T_2 are said to be the "sources" of the tuple (a, c) in the view, since deleting either of the tuples would lead to the deletion of (a, c) from the view. However, (b, c) is not a "clean source" of the tuple (a, c), because deleting (b, c) also leads to the deletion of the tuple (a', c) from the view. Only the tuple (a, b) is a "clean source" of the tuple (a, c). It follows that an exact translation of the deletion request is the deletion of (a, b) from the relation T_1. However, it would not be possible to find this translation if we adopted the view-dependency approach.

The view-dependency approach uses a graph, called view-dependency graph, to describe functional dependencies, keys and subset constraints. Based on the graph, the approach determines whether it is possible to find the "clean sources" of the tuple to be deleted from the view. Figure 2 shows the view-dependency graph of the view defined on $T_1(A, B)$ and $T_2(B, C)$. Due to space limit, we omit the detail of constructing the graph.

Given a view defined on a number of base relations, the necessary and sufficient condition for *any* instance of a base relation scheme to contain a "clean source" of a tuple to be deleted from the view is that for each attribute B of the view, there is an attribute A of the base relation schema such that there is a path from A to B in the view-dependency graph. This condition is independent of the state of the view and the state of the database (containing the base relations). Based on the condition, the conclusion is that an instance of $T_1(A, B)$ does not always contain a "clean source" of a tuple to be deleted from the view $\pi_{A,C}(T_1, T_2)$, because there is no path from the attributes of $T_1(A, B)$ to the attribute C of the view. Similarly, an instance of $T_2(B, C)$ does not always contain a "clean source" of a tuple to be deleted from the view $\pi_{A,C}(T_1, T_2)$, because there is no path because there is no path from the attributes of $T_2(B, C)$ to the attribute A of the view. The condition does not help us to see that, for the given instances of the relations $T_1(A, B)$ and $T_2(B, C)$ and the given view deletion request, there is indeed an exact translation of the deletion request, namely the deletion of (a, b) from the relation T_1.

Bancilhon and Spyratos [4] provides an abstract perspective on the view update problem. A view complement is defined as the supplement to a view definition covering all the information not covered by the view. The idea is to generate unique translation of view updates by telling the system which parts of the base relations should not be effected by the view updates. Gottlob et al [17] generalize the results of Bancilhon and Spyratos. However, there are two problems with the approach adopted in [4, 17]. First, the application of this idea for more concrete languages (e.g. SQL) remains largely unexplored. Secondly, the approach based on complementary views is apparently not practical due to its complexity. For example, it is shown that in the context of projective views over a single relation possibly having functional dependencies, finding a minimal complement is NP-complete [12].

A considerable amount of work has been focused on the role of database semantics in resolving ambiguity [13, 9, 24, 22, 25, 23]. Early approaches assume constraints on allowed data instances in form of dependency constraints [13, 9, 24]. Updates that violate these constraints should be rejected, thus to obtain the desired situation where exactly one method of updating stored data is possible. Keller [22, 23] claimed that for restricted types of project, select and join views, all the semantics required to resolve the ambiguity in the view update problem can be collected at view definition time. In many cases, however, it is impossible to determine the unique translation of an update request against a view unless there is some way to find out the user's intent. Masunaga [30] argued that for more general views, the collected semantics at view definition time are insufficient and ultimately the guidance of user intention is needed at view update request time to resolve the ambiguity. Masunaga suggested to associate solvers of semantic ambiguities with each operation of relational algebra used in an expression defining a view. In [31], the implications of view update policies are considered. A validation algorithm to check the correctness of view updates is proposed, with limitations on view definitions and the updating language. In [35, 29], it is maintained that both database semantics and application semantics are required; some can be obtained at view definition time, and the remaining must be obtained at view update request time.

Some work has been done on the view update problem in databases which allow null values [6, 7, 28, 2]. Langerak [28] studied the view update problem for a relational data model in which the base relation may contain (indexed) *nulls*. Langerak considered an abstract class of views which are total projections of the representative instances (the representative instances are defined to be the correct representation of all data in the database). Moreover, only independent database schemes are considered; that is, schemes for which global consistency is implied by local consistency. In this context, it is proven that the constant complement method from [4] is too restrictive to be useful. Structural properties of extension joins are derived that are important for understanding views. On the basis of these properties, minimal algorithms for translating a single view-tuple update are given. Vossen and Brosda [6, 7] studied updating though a universal schema interface. Combinations of attributes (”objects”) that may participate in an update operation have to be specified during the design phase of the database, and are then embodied into the database scheme by an extended synthesis algorithm. They form the basis for any insertion or deletion operation (modifications are not considered as a separate class of updates). Different from Langerak [28], Vossen and Brosda do not discreminate database nulls from nondistinguished variables.

In contrast to other published approaches, our approach can be used to take into account the question of whether a user is *authorized to update data* through a specific view. This question was brought up in [11]. However, the authorization question was not taken into account in the previous translation algorithms.

Finally, we mention also some recent work on specific versions of the view update problem. Langerak [28] studied the view update problem for a relational data model in which the base relation may contain (indexed) *nulls*. Langerak considered an abstract class of views which are total projections of the representative instances (the representative instances are defined to be the correct representation of all data in the database). Moreover, only independent database schemes are considered; that is, schemes for which global consistency is implied by local consistency. In this context, it is proven that the view complement approach from [4] is too restrictive to be useful. Structural properties of extension joins are derived that are important for understanding views. On the basis of these properties, minimal algorithms for translating a single view-tuple update are given. Vossen and Brosda [6, 7] studied updating though a universal schema interface. Combinations of attributes (”objects”) that may participate in an update operation have to be specified during the design phase of the database, and are then embodied into the database scheme by an extended synthesis algorithm. They form the basis for any insertion or deletion operation (modifications are not considered as a separate class of updates). Different from Langerak [28], Vossen and Brosda do not discreminate database nulls from nondistinguished variables. The view update problems in deductive databases are considered in [3, 2, 8, 14, 15, 32, 21, 33].

8 Summary

In this paper, we study a new approach to view update translation. The basic idea is to transform the problem into constraint satisfaction problems. We provide sound and complete transformation for updates of PSUJ views. This is, to our knowledge, the first reported attempt in this direction. The formal device we use is conditional table described in [20]. The approach has the following features compared with the previous approaches:

- The approach is declarative, which makes it possible to modify the problem formalization without changing the underlying problem solver. For instance, one can add or change semantic constraints on the database by modifying the CSPs; the underlying CSP solvers remain the same. On the other hand, it is possible to make use of the rich results of the CSP research to design algorithms for solving the view update translation problems.
- The approach is dependent of the current states of the database and the view. Thus it is better applied in view update time, rather than in view definition time.

The proposed approach should be adopted as a complement to the previous approaches to the translations of view updates. It can be used to solve those problems where the view updates do not satisfy the sufficient conditions for exact and unique translations [13], and where suggestions for possible translations of the view updates are highly desirable.

References

[1] S. Abiteboul and V. Vianu. Equivalence and optimization of relational transactions. *Journal of the ACM*, 35(1):130–155, January 1988.

[2] P. Atzeni and R. Torlone. Updating intensional predicates in datalog. *Data & Knowledge Engineering*, 8(1):1–17, April 1992.

[3] Paolo Atzeni and Riccardo Torlone. Solving ambiguities in updating deductive databases. In B. Thalheim, J. Demetrovics, and H.-D. Gerhardt, editors, *MFDBS 91 - 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems*, number 495 in Lecture Notes in Computer Science, pages 104 – 118, Berlin, 1991. Springer-Verlag.

[4] Francoise Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[5] Michael L. Brodie. On modelling behavioural semantics of databases. In Carlo Zaniolo and Claude Delobel, editors, *Proceedings of the 7th International Conference on Very Large Data Bases - 1981*, pages 32–42, Los Altos, CA, 1981. IEEE Computer Society Press.

[6] V. Brosda and G. Vossen. Updating a relational database through a universal schema interface. In ACM, editor, *4th ACM Symposium on Principles of Database Systems*, pages 66–75, New York, 1985. ACM Press.

[7] Volkert Brosda and Gottfried Vossen. Update and retrieval in a relational database through a universal schema interface. *ACM Transactions on Database Systems*, 13(4):449–485, December 1988.

[8] Francois Bry. Intensional updates: Abduction via deduction. In David H. D. Warren and Peter Szeredi, editors, *Logic Programming - Proceedings of the Seventh International Conference*, Logic Programming Series Research Reports and Notes, pages 561–575, Cambridge, MA, 1990. The MIT Press.

[9] C. R. Carlson and A. K. Arora. The updatability of relational views based on functional dependencies. In *3rd International Computer Software and Applications Conference*, Chicago Ill., November 1979. IEEE Computer Society.

[10] Stefano Ceri and Jennifer Widom. Deriving production rules for incremental view maintenance. In Guy M. Lohmann, Amilcar Sernadas, and Rafael Camps, editors, *Proceedings of the 17th International Conference on Very Large Data Bases - 1991*, pages 577–589, Palo Alto, CA, 1991. Morgan Kaufmann Publishers.

[11] Edgar F. Codd. *The Relational Model for Database Management, Vers. 2.* Addison-Wesley, Reading, MA, 1990.

[12] Stavros S. Cosmadakis and Christos H. Papadimitriou. Updates of relational views. *Journal of the ACM*, 31(4):142–160, 1984.

[13] Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 7(3):381–416, 1982.

[14] Hendrik Decker. Drawing updates from derivations. In S. Abiteboul and P. C. Kanellakis, editors, *ICDT'90 — Third International Conference on Database Theory*, number 470 in Lecture Notes in Computer Science, pages 437 – 451, Berlin, 1990. Springer-Verlag.

[15] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *Proceedings of the 2nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems - 1983*, pages 352–365, New York, 1983. ACM Press.

[16] Antonio L. Furtado and Marco A. Casanova. Updating relational views. In Won Kim, David S. Reiner, and Don S. Batory, editors, *Query Processing in Database Systems*, pages 127–144. Springer-Verlag, Berlin, 1985.

[17] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, December 1988.

[18] G. Grahne. *The Problem of Incomplete Information in Relational Databases.* Springer-Verlag, 1991.

[19] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *IEEE Database Engineering Bulletin*, 18(2):3–18, 1995.

[20] Tomasz Imielinski and Witold Jr. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, October 1984.

[21] A.C. Kakas and P. Mancarella. Database updates through abduction. In Dennis McLeod, Ron Sacks-Davis, and Hans Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases - 1990*, pages 650 – 661, Palo Alto, CA, 1990. Morgan Kaufmann Publishers.

[22] A. M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In ACM, editor, *4th ACM Symposium on Principles of Database Systems*, pages 154–163, New York, 1985. ACM Press.

[23] A. M. Keller. The role of semantics in translating view updates. *IEEE Computer*, 19(1):63–73, January 1986.

[24] A.M. Keller. Updates to relational databases through views involving joins. In P. Scheuermann, editor, *Improving Database Usability and Responsiveness, Proceedings of the 2rd Intl. Conf. on Databases*, pages 363–384. Academic Press, 1982.

[25] A.M. Keller. Choosing a view update translation at view definition time. In W. Chu, G. Gardarin, and S. Ohsuga, editors, *Proceedings of the 12th International Conference on Very Large Data Bases - 1986*, pages 467–474. Morgan Kaufmann Publishers, 1986.

[26] Udo Kelter. *Parallele Transaktionen in Datenbanksystemen.* B.I.-Wissenschaftsverlag, Mannheim, 1985.

[27] S. Kleene. *Introduction to Metamathematics.* Van Nostrand, Princeton, 1952.

[28] R. Langerak. View updates in relational databases with an independent scheme. *ACM Transactions on database systems*, 15(1):40–66, 1990.

[29] James A. Larson and Amit P. Sheth. Updating relational views using knowledge at view definition and view update time. *Information Systems*, 16(2):145–168, 1991.

[30] Y. Masunaga. A relational database view update translation mechanism. In U. Dayal, G. Schlageter, and L. Seng, editors, *Proceedings of the 10th International Conference on Very Large Data Bases - 1984*, pages 309–317, Palo Alto, CA, August 1984. Morgan Kaufmann Publishers.

[31] C. Medeiros and F. Tompa. Understanding the implications of view update policies. In A. Pirotte and Y. Vassiliou, editors, *Proceedings of the 11th International Conference on Very Large Data Bases - 1985.* Morgan Kaufmann Publishers, 1985.

[32] John rant, John Horty, Jorge Lobo, and Jack Minker. Updates in stratified disjunctive databases. *Journal of Automated Reasoning*, 11(2):249–268, October 1993.

[33] F. Rossi and S. Naqvi. Contributions to the view update problem. In G. Levi and M. Martelli, editors, *Logic Programming - Proceedings of the Sixth International Conference on Logic Programming*, pages 398–415. MIT Press, June 1989.

[34] L. Rowe and K. A. Schoens. Data abstractions, views and updates in rigel. In P. Bernstein, editor, *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, pages 71–81, New York, 1979. ACM.

[35] Amit P. Sheth, James A. Larson, and Evan Watkins. Tailor, a tool for updating views. In J. W. Schmidt, S. Ceri, and M. Missikoff, editors, *Advances in Database Technology — EDBT '88*, number 303 in Lecture Notes in Computer Science, pages 190–213, Berlin, 1988. Springer-Verlag.

[36] Michael R. Stonebraker. Implementation of integrity constraints and views by query modification. In W. F. King, editor, *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, pages 65–78. ACM SIGMOD, 1975.

[37] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, UK, 1993.

[38] L. Tucherman, Antonio L. Furtado, and Marco A. Casanova. A pragmatic approach to structured database design. In M. Schkolnick and C. Thanos, editors, *Proceedings of the 9th International Conference on Very Large Data Bases - 1983*, pages 219–231. Morgan Kaufmann Publishers, 1983.