

Updating Relational Views

Anthony L. Furtado and Marco A. Casanova

ABSTRACT

A view is a logical subset of the data base conceptual schema. Providing a facility for supporting views simplifies the user interface, but creates the problem of translating updates on views into equivalent updates on the data base. The translation of a view update may not be unique, it may not even exist or be ill-defined, and it may create inconsistencies in the data base or have side effects on the view. This paper surveys the two basic approaches proposed to solve the view update problem. The first approach suggests treating views as abstract data types so that the definition of a view includes all permissible view updates, together with their translations. The second approach leads to general view update translators and is based either on an analysis of the conceptual schema dependencies or on the concept of view complement to disambiguate view update translations.

1. INTRODUCTION

The complete logical structure of a data base is described by a *conceptual schema* listing the data structures and the integrity constraints of the data base [TSIC77,DATE81]. To delimit or focus on just those aspects of the data base relevant to each group of users, most systems also support the concept of an *external schema* or a *view*. Views simplify the user interface by allowing a user to ignore data that are not relevant to his application. They also provide a degree of protection by shielding data that should not be accessible by a class of users. Furthermore, views also help achieve a degree of logical data independence since it is often possible to alter the conceptual schema without changing the view.

We shall adopt in this paper the relational model of data [DATE81]. In this context a view is sometimes limited to a single-relation external schema and it can be described just like a query (as in System R [DATE81]). However, we shall not make this simplification here when dealing with the general theory of views. The view definition is stored by the system and can be used to materialize *derived relations* from the *stored relations* of the data base. A view definition facility is part of many relational data base systems [BROD82] such as SQL/DS [IBM], QBE [ZLOO77] and INGRES [STON76].

Users interact with a view by issuing queries and update requests. Queries present no particular problem. A query on a view is processed by composing the user query with the view definition, thus obtaining an equivalent query on the stored relations [STON75]. Conceptually, this process is equivalent to creating the derived relations first and then issuing the (view) query on these relations.

Processing view updates poses a difficult problem, though. To translate a view update u , we must find a data base update t such that t maps the current data base state s into a new state s' iff u maps the current view state v (generated from s) into a new view state v' , which is generated from s' . Now, t may not be unique, it may not even exist or be ill-defined, and it may create inconsistencies in the data base or have side effects on the view. It should be no surprise then that most data base management systems severely restrict the types of updates allowed on views. The *view update problem* refers to the question of choosing or defining correct view update translations.

This paper surveys several solutions proposed to the view update problem and briefly indicates how they could be used to enhance the view update facilities currently offered by data base management systems.

There are two basic approaches to the view update problem. One approach suggests to treat a view as an abstract data type [BROD81]. That is, the view definition should contain not only a description of how to derive view data from the underlying data base, but also a description of the set of allowed view updates together with their translations into updates over the conceptual schema [CLEM78, ROWE79, SEVC78, TUCH83]. This approach requires the design of an extended view definition language and methods of verifying that update translations are indeed correct (according to some clearly defined correctness criteria). This approach is addressed in Section 3 of the paper.

The second approach is to define general translation procedures. These procedures input a view definition, a view update, possibly with some extra information, and the current data base state and produce, if possible, a data base update that satisfies some desired properties [BANC81, CARL79, COSM83, KELL81, DAYA78, DAYA82, FURT79]. This solution is discussed in Section 4.

Both approaches need a theory characterizing precisely when a data base update is a correct translation of a view update [BANC81, CASA83, DAYA82]. An introduction to this theory, together with basic definitions and notation, is presented in the next section.

A word should be said about the style of presentation. The view update problem has some delicate aspects that cannot be covered without a certain amount of formalization. We have tried to keep the formalism to a minimum, however, and complement it with examples when the notation became too heavy. The space limitations also forced us to give a superficial treatment to some interesting approaches to the problem. The reader is thus urged to consult the references that generated each section.

2. AN INTRODUCTION TO THE VIEW UPDATE PROBLEM

2.1 Basic Definitions and Notation

This section gives general definitions for the notions of conceptual schema and view. Examples will be presented in Section 2.2.

A *relation scheme* is a statement of the form $R[A_1, \dots, A_n]$, where R is the *name* of the scheme and A_1, \dots, A_n are the *attributes* of the scheme; if the number of attributes is n , the scheme is said to be *n-ary*.

A *relational schema* is a pair $S = (RS, IC)$ where RS is a set of relation schemes and IC is a set of sentences, the *integrity constraints of the schema*. A *data base state*, or simply a *state*, of S is a function s assigning to each n -ary relation scheme in RS an n -ary relation over a given domain. A *data base update* of S is a mapping t from the set of states of RS onto itself. A state of S is said to be *consistent* iff it satisfies all integrity constraints of S . If an update t of S maps consistent states into consistent states, then we say that t *preserves consistency*.

An *external schema or view* over a data base schema S is a triple $V = (RV, CV, f)$ where $SV = (RV, CV)$ is a relational schema as above and f is a mapping from states of S into states of SV (over the same domain as the states of S). We also say that f is the *view definition mapping*, and we will be flexible about the way f is defined. A *view state* of V is simply a state of SV and a *view update* of V is just an update of SV .

This general definition of view will be used in sections 3 and 4.2. However, in sections 2.2, 4.1 and in parts of section 4.2, we will use a simplified concept of view.

When the view $V = (RV, CV, f)$ has a single relation scheme $R[A_1, \dots, A_n]$, no constraints, and the mapping is defined using a relational algebra or relational calculus expression, we will use the notation $R = f$ as an abbreviation for $V = (RV, CV, f)$ (the attributes of R are inherited from the expression).

Thrc
V'=(
state

In th
Q1.
Q2.
Q3.
Ans
Thu

In o
the
the
[DA
exe
leav
u(f(

To
view

2.2

Co
E'
cor
E:R
eve
mu
Co
opt
are

V1
V2
V3
V4
V5
V6

V'
V4
an

N.
th

C

Throughout this paper, $S=(RS,IC)$ will denote a data base schema, $V=(RV,CV,f)$ and $V'=(RV',CV',f')$ views over S ; s and s' will also be used to denote consistent states of S , v consistent states of V , u an update of V and t an update of S .

In this paper we will be mostly concerned with three questions [BANC81]:

- Q1. Given a view update u on V , when does a data base update t translate u ?
- Q2. What sets of view updates do we want to translate, that is, what sets of updates users are to be allowed on the view?
- Q3. How do we associate with each view update a data base update that translates it?

Answers to Q2 and Q3 reflect the view update strategy adopted and are dealt with in sections 3 and 4. Thus, we address only Q1 here.

In order for t to be considered a translation of u at all, t must take any s into some s' such that u takes the view state $f(s)$ into the view state $f(s')$. In addition, t must preserve consistency with respect to the integrity constraints of S . If t satisfies both conditions, we say that t is an *exact translation* of u [DAYA82]. But we might have exact translations that unnecessarily modify the data base, as exemplified in Section 2.2 below. So, we consider that t is (minimally) *acceptable* [BANC81] iff it leaves the data base state unchanged if u made no change on the view. That is, for any s , if $u(f(s))=f(s)$ then $t(s)=s$.

To summarize, any translation of a view update must be exact and acceptable. Broadly speaking, *the view update problem* is concerned with finding or defining such translations.

2.2. What are the Problems with the Execution of View Updates?

Consider a data base schema whose relation schemes are $E[EMP,DEP]$, $M[DEP,MGR]$ and $E'[EMP,DEP]$, noting that E and E' are defined on the same attributes (and so are trivially union-compatible [DATE81]). Assume that the constraints of the schema are two functional dependencies $E:EMP \rightarrow DEP$ and $M:DEP \rightarrow MGR$ as well as the inclusion dependency $E[DEP] \subseteq M[DEP]$. (That is, every EMP is assigned to a unique DEP in E , every DEP has a unique MGR in M , and every DEP in E must also occur in M). For simplicity, we shall consider that E' is not subjected to any constraint. Consider now a set of single-relation views whose mappings are defined through relational algebra operations on the stored relations. Using the simplified notation introduced in Section 2.1, the views are:

$V1 = E[DEP='Sales']$ (E restricted to $DEP = 'Sales'$)
 $V2 = M[DEP='Sales']$ (M restricted to $DEP = 'Sales'$)
 $V3 = E[EMP]$ (E projected on EMP)
 $V4 = E[DEP]$ (E projected on DEP)
 $V5 = E * M$ (the natural join of E and M)
 $V6 = E$ grouped by DEP and restricted to $count(EMP) > 2$
 $V7 = E \cup E'$ (E union E')
 $V8 = (E[DEP=DEP]M)[EMP,MGR]$ (E joined to M on DEP and projected on EMP,MGR)

Note: $V6$ is obtained by grouping (or partitioning, see [FURT77]) the tuples of E , so that employees of the same department are in the same group, and retaining only the groups of more than two tuples.

Consider now the following data base state:

E	EMP	DEP	M	DEP	MGR	E'	EMP	DEP
	John	Sales		Sales	Mary		Joan	Toys
	Peter	Finance		Finance	Jane			
	Alice	Sales						

together with the corresponding view states:

V1	EMP	DEP	V2	DEP	MGR	V3	EMP	
	John	Sales		Sales	Mary		John	
	Alice	Sales					Peter	
							Alice	
V4	DEP	V5	EMP	DEP	MGR	V6	EMP	DEP
	Sales		John	Sales	Mary			
	Finance		Peter	Finance	Jane			
			Alice	Sales	Mary			
V7	EMP	DEP	V8	EMP	MGR			
	John	Sales		John	Mary			
	Peter	Finance		Peter	Jane			
	Alice	Sales		Alice	Mary			
	Joan	Toys						

We shall use these views and the example data base state to illustrate several problems arising from the translation of view updates consisting of one-tuple insertions, deletions or replacements (following [FURT79]). Although more than one problem may be found in some examples, only one will be stressed in each case. We first state the general problem to be illustrated and then list one or more view updates, together with a brief discussion of what might happen when one tried to translate them. The data base state is assumed to be the one just described.

1) A constraint may be violated.

- insert (Peter,Sales) into V1:
The natural translation of this update, a simple insertion of (Peter,Sales) into E, violates the FD $E:EMP \rightarrow DEP$ since E contains (Peter, Finance) in the example data base state;
- delete (Sales, Mary) from V2:
Again, translating the update as a simple deletion of (Sales,Mary) from M violates $E[DEP] \subseteq M[DEP]$ since E contains (John,Sales) in the example data base state.

2) A result which does not conform with the view definition may be produced.

- insert (Mark,Finance) into V1:
V1 contains only employees in the Sales Department.

3) A result conforming with the view definition may be produced, but tuples lying outside the view may be involved.

- replace (Peter,Finance) by (Peter,Sales) in V1:
The straightforward translation of this view update as a replacement of (Peter,Finance) by (Peter,Sales) in E creates a tuple in V1 by modifying a tuple in E that originally generated no tuple in V1.
- delete (Peter,Finance) from V1:
As before, even if the tuple exists in E, it may not generate a tuple of V1.

4) An effect different from that expected from the nature of the operation may be produced.

- replace (John,Sales) by (John,Finance) in V1:
A straightforward translation as a replacement in E may be perceived as a deletion from V1; the modified tuple however remains in E.

5) It may

- inse:
The
valt

6) Attribu

- dele:
if t:
nan

7) Multiple

- repla:
To
(x,b

8) A multi

- repla:
The
tupl

9) An entir

- inser:
Con
(Jof
tran

10) More ti

- inser:
The

- repla:
Asst
repl

Observing
unsure abou

Case (1), fo
for an accep
changing th

Although w
instead exar
departments
execution of
this is but c
(4) and (6)

Interference
Thus, even i
sharing part

Case (5) is
and assignir
relational m
order of the

- 5) It may be necessary to include undefined values (nulls).
- insert (Roger) into V3:
The only reasonable translation is to insert (Roger,-) into E (where '-' stands for an undefined value).
- 6) Attributes outside the view may be affected.
- delete (Sales) from V4:
if this view update is translated as a deletion of all tuples of the form (x,Sales) from E, the names of the employees denoted by x are lost.
- 7) Multiple effects on the stored relations may be produced.
- replace (Sales) by (Marketing) in V4:
To translate this single-tuple update on V4, all tuples (x,Sales) in E must be changed to (x,Marketing).
- 8) A multiple effect on the view may be visible.
- replace (John,Sales,Mary) by (John,Sales,Jane) in V5:
The natural translation, a single replacement of (Sales,Mary) by (Sales,Jane) in M, changes all tuples (x,Sales,Mary) in V5 to (x,Sales,Jane), instead of only the intended tuple.
- 9) An entire group of tuples, or none at all, may be changed.
- insert (Roger,Sales) into V6:
Consider the translation that inserts the tuple into E. If there exists only two tuples, like (John,Sales) and (Alice,Sales) in E, then none may belong to V6 before the update; the translation may then result in the insertion of the entire group of three tuples into V6.
- 10) More than one translation may exist.
- insert (Roger,Sales) into V7.
The translation can insert the tuple either into E or into E' or into both.
 - replace (John,Mary) by (John,Jane) in V8.
Assuming that (John,x) is in E, and (x,Mary) and (y,Jane) are in M, the translation can either replace (John,x) by (John,y) in E or (x,Mary) by (x,Jane) in M.

Observing these examples, we remark that they have anomalous aspects and, hence, we would be unsure about how to translate them or if they should even be allowed.

Case (1), for instance, does not lead to an exact translation. Case (2) does not meet our requirement for an acceptable translation, since, if allowed, it would modify the stored relation E without actually changing the view V1.

Although we could point out non-exact or non-acceptable translations in the other cases, we shall instead examine other related problems. In the first example of case (3), if no user's view includes both departments involved in the attempted transfer of the employee, we could think of a "cooperative" execution of the update by two authorized users, which mimics a common real-world practice. In fact, this is but one example of the phenomenon of mutual *interference* (or *influence*) among views. Cases (4) and (6) also show updates having effects outside the view being directly updated.

Interference is really a much broader issue^{*} in data bases, since views need not be mutually disjoint. Thus, even if the effects of an update are apparently confined to one view, they may affect other views sharing part of the same stored relations. For instance, the update on V4 in case (7) also affects V1.

Case (5) is another example where cooperation is needed. It suggests that introducing an employee and assigning him to a department are separate actions, possibly by different users. Note that the relational model requirement that keys cannot have undefined values prevents us from inverting the order of the two actions: a tuple like (-,Sales) could not be inserted into E.

In general, introducing additional constraints can help disambiguate translations of view updates. In case (10), the first example dealing with the union of E and E' is ambiguous because there is nothing in their definition to distinguish them. We might suppose instead that E is constrained to record only full-time employees and E' only part-time employees, thereby introducing a distinguishing implicit attribute.

Similarly, the ambiguity of the second example of case (10) could be removed with an additional functional dependency $M:MGR \rightarrow DEP$; this would exclude the insertion of (x,Jane) into M if (y,Jane) is already in M.

The anomaly in case (4) is that the effect is incongruous with the nature of the intended operation. In cases (8) and (9) the expected effect is produced, but other effects also take place. This situation is not very different from interference among views; here the unsolicited side effects result from an operation on the same view but would appear to its user as unrelated to his action.

Combining multiple updates triggered by a single update and effects outside the view, we may envision a way to perform the update of the second example of case (1) that preserves the inclusion dependency. The deletion of (Sales, Mary) from M would be followed by the deletion of all tuples of the form (x,Sales) from E. However, this is again an ambiguous situation: how do we decide if the constraint must be enforced either by forbidding the offending update or by triggering other compensating updates? As before, the answer is to introduce further constraints, in this case on the relationship between employees and departments. Using CODASYL 78 [DATE81] terminology, we would decide for the former solution if employees are *mandatory* members and for the latter, if they are *fixed* members. So, an open-ended number of constraints, other than just dependencies, may be used to reduce the degree of arbitrariness when allowing or forbidding updates on views and choosing how to translate them.

3. THE ABSTRACT DATA TYPES APPROACH

The abstract data types approach characterizes the objects of a type by the operations performed on them and hides from the programmers details about how such objects are represented [BROD81].

A trivial example that will help understand how the approach can be applied to view updates is that of stacks. Suppose that stacks are represented by PASCAL arrays and that the stack operations CREATE, ISEMPY, TOP, PUSH, POP are implemented (translated) by functions or procedures in whose bodies assignment and subscripting operations on arrays are used. Thus, inside a procedure for the PUSH operation we might have:

```
...
t := t + 1;
A[t] := v;
...
```

where t indicates the top element of the stack represented by array A, and v is the value being pushed.

We would be grossly mistaken if we interpreted in terms of arrays what the procedure above does. As an array operation what we have is simply that an array element is assigned a value, whereas after the PUSH operation all elements of A with subscripts less than the incremented value of t are invisible. More precisely, this is a consequence of the combined behavior of the stack operations, which allows only the top element to be seen. Conversely, the POP operation, which does not even change the array (it only decrements t), not only removes the current top element but also restores the previous one.

The analogy with views is immediate. Stacks are comparable to derived relations, permitting us to see a limited part of the stored relations, which in turn are comparable to the arrays used to represent stacks. It follows that, analogous to stack operations (translated into one or more array operations), we should introduce *application-oriented operations* [SEVC78] to handle views of specific data bases (an example will be given at the end of this section). The view definition would specify, for each application-oriented operation, an exact translation on the underlying stored relations. Since the designer has complete control of the translations, they may possibly have multiple effects, all of which may not necessarily be visible through the view.

Some application-oriented operations pertinent to our running example are to hire, fire, assign and transfer employees, to create, rename and extinguish departments, to raise a department to the status of "major department" (more than two employees), to attach an employee to a manager, etc.

Some of the benefits of this approach are:

- 1) Certain updates are no longer anomalous when we cease to consider them as unconditional applications of single tuple insertion, deletion or replacement operations.
- 2) Ambiguity is avoided since, when explicitly translating the application-oriented operations, arbitrary decisions can be made in the absence of pertinent constraints.
- 3) The application-oriented operations will often provide a more user-friendly interface to non-professionals in data processing who are knowledgeable in their application areas.
- 4) Constraints are automatically enforced if view updating is restricted to the defined application-oriented operations. (Note that this strategy also enforces constraints more efficiently than most other automatic enforcement strategies). This decision corresponds to the *encapsulation* strategy, which is inherent in the abstract data types approach.

An unusual feature of data bases, as compared to the well-known cases of abstract data types in the programming languages literature, is that data bases are shared objects. However this does not make the approach inapplicable, even though it requires a careful analysis to ensure that the entire set of operations effectively excludes all undue interferences among the views (see [CASA83]).

A disadvantage of the approach is the work required to define the operations and the need to possibly redefine them during the lifetime of the data base. Yet, this seems preferable to the prevailing practice in many organizations of forbidding users to perform updates in real-time, out of fear of constraint violation. The problem is especially serious due to the fact that most data base management systems neither permit the declaration nor enforce constraints, except for the most trivial classes.

On the other hand, the abstract data type approach is compatible with existing data base management systems, as demonstrated in [TUCH83] for SQL/DS, although the introduction of additional language features seems desirable. To give the flavor of such features, we show an example of a module through which a user can attach an employee to a manager. (A module is understood here as a set of data base data structures plus a set of operations on these structures).

Example 3.1: Definition of a Module.

```

module SUBORDINATION extends PERSONNEL_DB
  schemes
    B[EMP,MGR]
  constraints
     $\forall e,m,n (B(e,m) \wedge B(e,n) \Rightarrow m=n)$ 
  operations
    ATTACH(e,m):
      insert (e,m) into B;
  using
    view definitions
      B[EMP,MGR]: { (e,m) /  $\exists d (E(e,d) \wedge M(d,m))$  }
    operations translations
      ATTACH(e,m):
        if  $\exists d M(d,m) \wedge \neg \exists d' E(e,d')$ 
        then insert (e,d) into E;
end-module

```

We assumed the existence of a previous module, PERSONNEL__DB, defining two relation schemes, E[EMP,DEP] and M[DEP,MGR], subjected to the integrity constraints $E:EMP \rightarrow DEP$, $M:DEP \rightarrow MGR$, $M:MGR \rightarrow DEP$ and $E[DEP] \subseteq M[DEP]$.

The module SUBORDINATION is then defined over PERSONNEL_DB. It contains a relation scheme, corresponding to a derived relation, and an integrity constraint, which is a consequence of the FDs of the PERSONNEL_DB module. The only operation is ATTACH. Users of the module need know only these three parts of the module definition: the relation schemes, integrity constraints and operations. The definitions under the 'using' clause should not be visible since they correspond to an implementation of the module objects in terms of the objects of the underlying module. Finally, note that the constraints of PERSONNEL_DB are preserved by the translation of the ATTACH operation, since it produces no effects if *m* does not manage a department or if *e* is already assigned to some department. Since the module constraint is a consequence of those of PERSONNEL_DB, the translation also indirectly preserves consistency with respect to the module constraints.

As an example of the effects of ATTACH, assume that the current data base state is

E	EMP	DEP	M	DEP	MGR
	John	Sales		Sales	Mary
	Peter	Finance		Finance	Jane
	Alice	Sales			

and the view state is

B	EMP	MGR
	John	Mary
	Peter	Jane
	Alice	Mary

Then, we will have

- ATTACH(John,Jane) fails since John has already been assigned to Sales.
- ATTACH(Robert,Laura) fails since Laura does not manage a department.
- ATTACH(Robert,Jane) succeeds since Jane manages a department and the tuple (Robert,Finance) is inserted into E. □

The module concept illustrated above is fully discussed in [TUCH83], where the combination of views as an encapsulation device with the modular design of data base schemas is investigated.

4. THE AUTOMATIC TRANSLATION APPROACH

The broad goal of the automatic translation approach to the view update problem is to define, for each view *V* and each *complete* set *U* of updates on *V*, a *translator* *T* for *U*.

A class *U* of updates users are allowed to make on a view *V* is said to be *complete* if it satisfies the following minimum requirements [BANC81,SEVC78]:

- 1) If *u* and *v* are in *U* then the composition of *u* and *v* is also in *U*.
- 2) If the view is updated, using an update in *U*, then there is an update in the set that brings the view back to the original state (thus users may cancel the effect of any previous update).

A mapping from view updates in *U* to data base updates is in turn said to be a *translator* for *U* iff [BANC81]

- 1) for any u in U , $T(u)$ must be an exact, acceptable translation of u (see Section 2.1), and
- 2) the translation of the composition of two view updates in U must be the composition of the translations.

Example 4.1:

Let S be a data base schema whose relation schemes are $E[EMP,DEP]$ and $M[DEP,MGR]$, and whose constraints are $E:EMP \rightarrow DEP$, $M:DEP \rightarrow MGR$. Let V be a view over S whose only scheme is $EN[EMP]$, defined by projecting E onto EMP .

Let U be the set of all single tuple insertions and deletions in V and their composition. Then one can easily prove that U is complete.

Let T be the translator that associates to each u in U a translation $T(u)$ defined as follows:

- 1) An insertion of a tuple (e) in EN is translated as an insertion of a tuple $(e,-)$ in E , provided that e does not occur in EN ;
- 2) A deletion of a tuple (e) from EN is translated as a deletion of a tuple (e,d) from E , for some department d .
- 3) The translation of a sequence of insertions and deletions in EN is translated as the sequence of translations of the insertions and deletions.

Then one can also easily prove that T is a translator for U . \square

The basic problems one must face when defining a translator is that the view definition alone may not provide sufficient information on the relationship between view updates and data base updates. A way out of this dilemma is to explore the data dependencies pertaining to the conceptual schema, together with the view definition, to detect conditions under which view updates can always be unique and correctly translated [CARL79,KELL81,DAYA78,DAYA82,FURT79]. This solution is discussed in Section 4.1. A second solution uses the concept of view complement to disambiguate the semantics of view updates and is addressed in Section 4.2.

4.1 The View Dependency Graph Approach

The first strategy to design general translators for view updates we consider explores the structure of the dependencies of the underlying conceptual schema to disambiguate view updates. We present a brief summary of this approach, following [DAYA78,DAYA82].

In this section we will work under the following assumptions:

- 1) The constraints of the conceptual schema are restricted to be single-attribute functional dependencies and keys, that is, FDs of the form $R:A \rightarrow B$, where A and B are attributes of R , and keys consisting of just one attribute. (This assumption is stronger than the one found in [DAYA82], but it simplifies the presentation of the basic ideas of the method). Moreover, every relation scheme must have a primary key.
- 2) Views are restricted to contain just one relation scheme and no constraints; moreover, the view mapping must be defined using just restrictions, selections and joins involving only equalities.
- 3) The view updates are restricted to be single- and multi-tuple deletions and replacements and single tuple insertions.

Due to the nature of the results, it is convenient to adopt a language called SQL [DATE81,IBM] to express view definition mappings, and also to express view updates. A single-relation view, with no constraints, will then be defined as follows:

```
(1) DEFINE VIEW R (B1,...,Bn)
    AS SELECT t1.Ai1,...,tin.Aijn
       FROM R1 t1,...,Rm tm
       WHERE <qual>
```

where <qual> contains only clauses of the form "t.A = c" or "t.A = u.B" (t and u may be the same). We say that the attribute A_{ijk} of R_i *generates* the view attribute B_k, for each k in [1,n].

We shall also assume for simplicity that no two tuple variables range over the same scheme and that, if the qualification implies an equality of the form t.A=u.B, then this equality is explicitly included in the qualification.

The structure of the view definition is captured by defining a labeled directed graph $G(V)=(N,E)$ constructed as follows:

- 1) for each attribute A of each relation name R_i occurring in the FROM clause, there is a node in N labeled "R_i.A";
- 2) for each view attribute B_j, there is a node labeled "R.B_j";
- 3) for each constant c occurring in the qualification, there is a node labeled "c" in N;
- 4) for each view attribute B_k, there are arcs (R.B_k,R_i.A_{ijk}) and (R_i.A_{ijk},R.B_k), if A_{ijk} generates B_k;
- 5) for each equality R'.A'=R".A" in the qualification, there are arcs (R'.A',R".A") and (R".A",R'.A');
- 6) for each restriction R'.A'=c in the qualification, there are arcs (R'.A',c) and (c,R'.A').

The *view dependency graph* is obtained by incorporating in $G(V)$ the information provided by the FDs of the underlying schema. This is done as follows:

- 1) for each FD $R_i:A \rightarrow B$, add the arc (R_i.A,R_i.B) to $G(V)$;
- 2) if the qualification implies R'.A'=c, which can be detected by a path from R'.A' to c in $G(V)$, add arcs (R".A",R'.A') to $G(V)$, for each node R".A" of $G(V)$ such that R" occurs in the FROM clause.

Note that the view dependency graph depends only on the view definition and on the dependencies of the conceptual schema: Hence it can be constructed and stored together with the view definition.

Example 4.2:

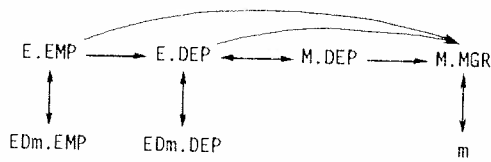
Let S and V be defined as follows:

a) S is a data base schema whose relation schemes are E[EMP,DEP] and M[DEP,MGR], and whose constraints are $E:EMP \rightarrow DEP$, $M:DEP \rightarrow MGR$.

b) V is a view over S defined as:

```
DEFINE VIEW EDm(EMP,DEP)
  AS SELECT E.EMP, E.DEP
     FROM E, M
     WHERE E.DEP=M.DEP AND M.MGR=m
```

Then, the view dependency graph of V is:



□

We shall now state how the view dependency graph can be used to derive conditions on the translatability of deletions. Similar conditions can be found in [DAYA82] for insertions and replacements.

Let V be a view defined as in (1) above. Again using the syntax of SQL, a *simple deletion* on V is a deletion of the form:

```
(2) DELETE R
      WHERE Bk1=c1 AND ... AND Bkp=cp
```

Consider the following translation procedure for deletions of this form:

step 1:

Choose one relation name R_i occurring in the FROM clause of V .

step 2:

The translation of the deletion will be

```
DELETE Ri
  WHERE Ri.Ki IN
    ( SELECT ti.Ki
      FROM R1 t1, ..., Rm tm
      WHERE <qual>
        AND SOURCE(Bk1)=c1 AND ... AND SOURCE(Bkp)=cp )
```

where K_i is the primary key of R_i (which exists by assumption), and $SOURCE(B_j)$ denotes the expression $R_q.A_{iq}$, if A_{iq} generates B_j .

It is obvious that the proposed translation, call it t , preserves consistency since deletions cannot violate keys and FDs. Moreover, t is clearly acceptable since, if the view deletion qualification fails, so does the qualification of t . Hence, if the view deletion does not modify the view, t also does not modify the data base. It can also be easily shown that t performs the desired deletions, that is, the tuples of V that should be deleted indeed are. However, t may have side effects and, hence, may not be an exact translation. The view dependency graph then comes in our help. It is proved in [DAYA82] that t is an exact translation of u iff, for each attribute B of V , there is an attribute A of the relation scheme R_i chosen in the first step of the procedure such that there is a path from A to B in the view dependency graph.

The following example illustrates a successful application of this result.

Example 4.3:

Let S and V be as in Example 4.2. Let u be the following simple deletion on V

```
DELETE EDm
  WHERE DEP=d
```

A possible translation t of u is:

```
DELETE E
  WHERE E.EMP IN
    ( SELECT E.EMP
      FROM E, M
      WHERE E.DEP=M.DEP AND M.MGR=m
        AND E.DEP=d )
```

Since there is a path from E.EMP to EDm.EMP and a path from E.DEP to EDm.DEP, t is an exact translation of u .

As a concrete scenario, assume that $m='Mary'$ and $d='Sales'$ and let the current data base state and the current view state for ED-Mary be

E	EMP	DEP	M	DEP	MGR	ED-Mary	EMP	DEP
	John	Sales		Sales	Mary	John	Sales	
	Peter	Finance		Finance	Jane	Alice	Sales	
	Alice	Sales		Toys	Mary	Albert	Toys	
	Albert	Toys						

Then, the translation of the previous view deletion (for $m='Mary'$ and $d='Sales'$) will correctly delete tuples (John,Sales) and (Alice,Sales) from E. Furthermore, note that if we change d to be 'Finance', the translation will not delete (Peter,Finance) from E since (Finance,Mary) is not in M (hence the qualification of the translation fails). \square

The next example illustrates an unsuccessful translation.

Example 4.4:

Let S , V and u be as in Example 4.3. Consider the following translation t' for u

```
DELETE M
WHERE M.DEP IN
( SELECT M.DEP
  FROM E, M
  WHERE E.DEP=M.DEP AND M.MGR=m
    AND E.DEP=d )
```

This time we cannot deduce that t' is an exact translation of u since there is no path from an attribute of M to EDm.EMP.

Indeed, assuming the same concrete scenario as in Example 4.3, the translation t' will delete (Sales,Mary) from M. As a consequence we have:

- (John,Sales) and (Alice,Sales) vanish from the view ED-Mary, but remain in E.
- (Sales,Mary) is deleted from M, which is by itself undesirable since the intention of the view update was to delete all employees of the Sales department, not its manager.
- If (Sales,Mary) is reinserted into M, the two tuples (John,Sales) and (Alice,Sales) will reappear in ED-Mary. Moreover, if (Sales,Jane), say, is inserted into M, these two tuples will appear in ED-Jane. \square

The procedure described above for translating simple deletions is not deterministic since there might be more than one R_i satisfying the above conditions. The view definition should then include additional information to completely disambiguate the translation (see the following Section).

To conclude, we observe that similar, but somewhat more complex conditions, are obtained in [DAYA82] for insertions and replacements. The net result is a translator for a large class of (sequences of) insertions, deletions and replacements, which can be used to relax the very stringent conditions imposed on view updates by some currently existing systems (such as System R [DATE81]).

4.2 The View Complement Approach

A second strategy to completely specify the translation of view updates is to tell the system which parts of the data base should not be affected by the view updates. We will present a brief summary of

the theory behind this approach, taken from [BANC81], and some suggestions to transform it into a practical tool [CHAN83,COSM83].

Let us begin with the key definition of view complement. Let $S=(RS,IC)$, $V=(RV,CV,f)$ and $V'=(RV',CV',f')$ be as in Section 2.1. Intuitively, V' is the complement of V iff V' , together with V , suffices to reconstruct the original data base. That is, any information not contained in V should be present in V' . More precisely, let $f \times f'$ denote the function f'' such that $f''(s)=(f(s),f'(s))$. Then, V' is the complement of V iff the mapping $f''=f \times f'$ is one-to-one when restricted to the consistent states of S . This implies that we can move from a consistent state s of S to a pair of states $(v,v')=f''(s)$ of V and V' , and then return to s (via the inverse of f'') without losing information.

Example 4.5 [BANC81]:

If s denotes a state and E a relational expression, let $s(E)$ denote the value of E in s . Let S , V and V' be as follows:

a) S is a data base schema whose relation schemes are $E[EMP,DEP]$ and $M[DEP,MGR]$, and whose constraints are $E:EMP \rightarrow DEP$, $M:DEP \rightarrow MGR$, $M:MGR \rightarrow DEP$, and $E[DEP] \subseteq M[DEP]$ and $M[DEP] \subseteq E[DEP]$.

b) V is a view over S whose only relation scheme is $EM[EMP,MGR]$, with $EM:EMP \rightarrow MGR$ as constraint, and whose view mapping is $f = (E * M)[EMP,MGR]$ (the natural join of E to M , followed by the projection on EMP and MGR). That is, $f(s)=v$ iff $v(EM)=s((E * M)[EMP,MGR])$

c) V' is another view whose only relation scheme is $M'[DEP,MGR]$, with $M':DEP \rightarrow MGR$ and $M':MGR \rightarrow DEP$ as constraints, and whose view mapping is $f' = M$ (that is, M' coincides with the manager table M itself). Hence, we have $f'(s)=v'$ iff $v'(M)=s(M)$.

Then V' is the complement of V . Indeed, let $f''=f \times f'$. Recall that f'' maps a state of S into a pair of states of V and V' . Recall that a state of S is a function assigning to E and M binary relations, a state of V is a function assigning to EM a binary relation, and a state of V' is a function assigning to M' another binary relation. To prove that V' is the complement of V , we have to show that f'' is one-to-one when restricted to the consistent states of S . We proceed as follows

- (1) for any state s of S , $f''(s) = (v,v')$ iff
 $v(EM) = s((E * M)[EMP,MGR])$ and
 $v'(M') = s(M)$

Now, using the dependencies of S , we can show that

- (2) for any consistent state s of S ,
 $s(((E * M)[EMP,MGR]) * M)[EMP,DEP] = s(E)$

Let s' and s'' be two consistent states of S and suppose that $f''(s')=f''(s'')$. That is, using (1), we have

- (3) $s'((E * M)[EMP,MGR]) = s''((E * M)[EMP,MGR])$

- (4) $s'(M) = s''(M)$

Using (2), (3) and (4) above, it follows that

- (5) $s'(E) = s'(((E * M)[EMP,MGR]) * M)[EMP,DEP]$
 $= s''(((E * M)[EMP,MGR]) * M)[EMP,DEP]$
 $= s''(E)$

Thus, from (4) and (5) it follows that $s'=s''$, which establishes that f'' is one-to-one on the consistent states of S . \square

The above example illustrates that it is difficult to prove that two views are complementary. However, it is relatively straightforward to convince oneself that (see [BANC81])

- 1) A view always has a complement (the underlying schema is always a complement of any view);
- 2) a view can have many complements, even if only minimal complements are considered (under a precise notion of minimality); and
- 3) the complement of a view describes the information not visible within the view.

It is a remarkable result of the theory that if u is at all translatable, then its translation can be easily computed.

More precisely, we say that u is V' -translatable iff

- 1) for any consistent state s of S there is a consistent state s' of S such that $f(s')=u(f(s))$ (i.e., s' reflects the update u on V); and
- 2) $f'(s')=f'(s)$ (the value of the complement V' is the same in s and s').

If g and h are functions, let $g \circ h$ denote the composition of g and h , that is, $(g \circ h)(x)=g(h(x))$. Let I denote the identity function on the set of states of V' . If u is V' -translatable, then the translation of u is achieved by the update t on S defined by (see Figure 1)

$$t = g \circ h \circ f''$$

where

- $f'' = f \times f'$,
- g is the inverse of f''
- $h = u \times I$.

We call the mapping T associating each update u with the translation t as defined above the *translator induced by V'* .

The theory of view complement then suggests the following strategy to cope with view updates. The data base administrator, together with the definition of each view V , must also fix a complement V' of V indicating which parts of the data base should not be affected by updates on V . A translation procedure must then be found such that each update u on V is either rejected as untranslatable or mapped to a data base update t that maintains the data in V' invariant. If U is a set of updates on V such that each update in U is V' -translatable, then it follows that the translator T induced by V' is exactly the desired procedure.

But the converse also holds. If T is a translator for a given complete class U of updates on V , then there is a complement V' of V such that (i) any u in U is V' -translatable; and (ii) T is the translator induced by V' [BANC81].

To render the above strategy usable in practice, we must find

- 1) a method of checking if V' is the complement of V ;
- 2) a method of deciding if each update in U is V' -translatable; and

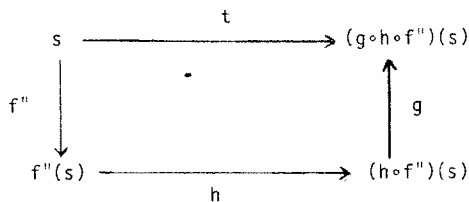


Figure 1: Translation of View Update u

3
F
T
s
v
L
o
It
I:
d
s
s:
ti
N
ro
o
ti
1
2
3
N
E
A
br
1)
2)

- 3) a method of computing the inverse of $f \circ X \circ f'$, which is necessary to completely describe the translator.

Further work is still required in these directions, but some results already exist.

These three questions have been investigated in [COSM83] for the simple case where S contains a single relation scheme $R[A_1, \dots, A_n]$, and a set F of functional and join dependencies, and where all views are single-relation and are defined by projections of R .

Let $ATTR = \{A_1, \dots, A_n\}$. Using our simplified notation, let $V = R[X]$ and $V' = R[X']$ be two views obtained by projecting R on X and on X' , respectively.

It is shown that V' is a complement of V iff $ATTR = X \cup X'$ and F implies the MVD $R: X \cap X' \twoheadrightarrow X' - X$. In one direction, this result is obvious, since if F implies the MVD $R: X \cap X' \twoheadrightarrow X' - X$, then the decomposition of R into $R[X]$ and $R[X']$ is lossless. Therefore, if s and s' are two consistent states of S such that $s(R[X]) = s'(R[X])$ and $s(R[X']) = s'(R[X'])$, we have $s(R) = s(R[X] * R[X']) = s'(R[X] * R[X']) = s'(R)$, which suffices to establish that V and V' are complementary.

Necessary and sufficient conditions for the translatability of single-tuple insertions, deletions and replacements, with emphasis on the complexity of the tests involved, are also presented. As an example of the kind of results obtained, suppose now that all constraints are FDs. The insertion of a single tuple u into V is translatable as the insertion of a tuple t such that $t[X] = u$ and $t[X']$ iff

- 1) F implies $R: X \cap X' \twoheadrightarrow X'$, but not $R: X \cap X' \twoheadrightarrow X$;
- 2) $u[X \cap X']$ is in $R[X \cap X']$; and
- 3) the insertion of T into R does not violate any of the dependencies in F .

Note that, since F implies $R: X \cap X' \twoheadrightarrow X'$, t is uniquely defined.

Example 4.6:

Let S be a conceptual schema whose only relation scheme is $E[EMP, DEP, PROJ]$ and whose only constraint is $E: EMP \twoheadrightarrow DEP$. Let V be the view defined by projecting E onto EMP and $PROJ$. Then the view V' obtained by projecting E onto EMP and DEP is a complement of V by the above result. Moreover, the insertion of $\langle e, p \rangle$ into V can be translated as the insertion into E of a tuple of the form $\langle e, d, p \rangle$ such that $\langle e, d \rangle$ is in $E[EMP, DEP]$, provided conditions (2) and (3) are also satisfied (both depend on the current state of S). Note that condition (1) is trivially satisfied.

For example, consider the following data base state

R	EMP	DEP	PROJ
	John	Sales	P1
	Peter	Finance	P1
	Albert	Sales	P2

Then, the insertion of $u = (\text{John}, P2)$ is correctly translated by $t = (\text{John}, \text{Sales}, P2)$ since none of the conditions are violated. But the insertion $u = (\text{Robert}, P2)$ fails since condition 2 is not met.

An interesting suggestion to render the idea of view complement usable in practice, which we now briefly describe, can be found in [CHAN83]. We will work under the following assumptions:

- 1) Updates are restricted to be single- or multi-tuple insertions, deletions and replacements.
- 2) Only *simple* views are considered, that is, views consisting of a single relation scheme, with no constraints, whose view mapping is defined by using a single relational operator. Thus, a view can be defined using the simplified notation $V = e$, where e is a relational expression with a single relational operator.

Since views over views are allowed, this last restriction can be circumvented by breaking the definition of a complex view into a sequence of definitions of simple views. On the other hand, this restriction induces a taxonomy of views according to which relational algebra operator was used in the view mapping (an approach also taken in [FURT79]).

The general idea is to construct an interpreter that takes a given view update u , the view definition and the current data base state and produces a unique exact, acceptable translation of u . One of the problems again is ambiguity. Instead of using view complements directly, a *relational invariant* is defined for each type of update on each view. A relational invariant is a relational expression indicating that no update of the type in question must neither alter any tuple in the relation denoted by the invariant nor insert or delete any tuple that satisfies the invariant.

For example, let $V=A \cup B$ be a single-relation view obtained by the union of the base relations A and B . Consider an insertion u on V . Then, we can translate u either as an insertion on A or an insertion on B or both. If, however, we declare A (understood as an atomic relational expression) as an invariant under insertions on V , the ambiguity disappears.

Now, given an update u on a view V , the interpreter constructs all possible translations for u . This is accomplished with the help of *update templates* indicating all possible choices for each view update. The interpretation is a recursive process since V may be defined over other views. Those candidate translations that violate consistency or violate an invariant are discarded. If only equivalent translations result from the interpretation process, the view update is accepted as translatable and the operations resulting from the translation process are executed (see [CHAN83] for details).

It is feasible to implement such translation strategy, provided that violations of invariants or integrity constraints can be detected with reasonable efficiency. A way to achieve this goal would be to restrict the type of invariants or constraints allowed by the system.

5. CONCLUSIONS

A common tendency is to define views only for query and authorization purposes, and afterwards try to solve the problems created by view updates. We feel that the update requirements should be considered from the start.

From the point of view of the data base designer, the automatic translation approach is on a first thought much more comfortable. However, if additional information is not included in the view definition, the class of view updates that can be correctly translated by the system is extremely limited. Moreover, this additional information, for example in the form of adequate view complements, is not always easy to define since it involves several checks to guarantee that indeed view update translations can be correctly performed.

The abstract data types approach, on the other hand, places the burden of implementing view updates on the data base designer right from the beginning. He must define a set of application-oriented operations that represent the desired updates, together with translations that perform the operations and preserve consistency of the data base.

As our last remark, future data base systems should include better view definition facilities that take into consideration the view update problem. As a humble start, a view definition language following the abstract data types approach could be added to existing systems. Using results from the automatic translation approach, view update translators could also be designed to handle a richer class of view updates than that processed by currently existing systems. Furthermore, such results can be combined with the abstract data type approach to create comprehensive view definition software tools that alleviate the burden of the data base designer. Also relevant in this regard are efforts towards the synthesis of update transactions with the help of theorem provers [SANT81].

References

- (BANC81) (BROD81) (BROD82) (CARL79) (CASA83) (CHAN83, see CHAN83b)
 (CLEM78) (COSM83) (DATE81, see DATE82) (DAYA78) (DAYA82, see DAYA82e)
 (FURT77) (FURT79) (IBM) (KELL81) (ROWE79) (SEVC78) (SANT81) (STON75)
 (STON76) (TUCH83) (TSIC77, see TSIC77a) (ZLOO77)