Update Semantics of Relational Views

F. BANCILHON and N. SPYRATOS
INRIA, France

A database view is a portion of the data structured in a way suitable to a specific application. Updates on views must be translated into updates on the underlying database. This paper studies the translation process in the relational model.

The procedure is as follows: first, a "complete" set of updates is defined such that

(i) together with every update the set contains a "return" update, that is, one that brings the view back to the original state;
(ii) given two updates in the set, their composition is also in the set.

To translate a complete set, we define a mapping called a "translator," that associates with each view update a unique database update called a "translation." The constraint on a translation is to take the database to a state mapping onto the updated view. The constraint on the translator is to be a morphism.

We propose a method for defining translators. Together with the user-defined view, we define a "complementary" view such that the database could be computed from the view and its complement. We show that a view can have many different complements and that the choice of a complement determines an update policy. Thus, we fix a view complement and we define the translation of a given view update in such a way that the complement remains invariant ("translation under constant complement"). The main result of the paper states that, given a complete set U of view updates, U has a translator if and only if U is translatable under constant complement.

Key Words and Phrases: relation, relational model database, data model, data semantics, conceptual model, database view, view updating, update translation
CR Categories: 3.70, 4.33, 4.34

1. INTRODUCTION

In database systems the amount of data to be stored and its structure are decided by the database administrator. Individual users, however, may be interested in just a portion of the data and would certainly like to see it structured in a way suitable to their specific application. The concept of a user's view of the database was specifically introduced to satisfy these requirements. A view is a query definition named and stored. That is, a view is a dynamic picture of a query. INGRES [7] and system R [2] are examples of systems that provide the view facility. In a sense, such a facility allows the user to define a database whose states depend on the underlying database. This dependence is expressed in the view definition mapping which associates with each database state a view state.

Authors' address: Institut National de Recherche en Informatique et en Automatique, B.P. 105, 78150 Le Chesnay, France.

The user interacts with his view by issuing queries and update requests. Queries do not present particular problems. The view definition mapping is sufficient to translate view queries into database queries. View updates, however, present a difficult problem: A database update that translates a view update must take the database to a state mapping onto the updated view. Now there is, in general, more than one database update that satisfies this requirement. The problem is how to choose one: that is, how to choose a unique operation which can be applied to the underlying base relations and which will result in exactly the specified changes to the user's view.

The objective of this paper is to study the view update translation in the context of the relational model. No attempt is made to produce computational algorithms. We proceed as follows: After a review of the relational model, we present formal definitions for updates and views. We then define "complete" sets of view updates so that they reflect users' requirements. A complete set is such that

(i) the composition of two updates from the set is also in the set;
(ii) if the view is updated, using an update from the set, then there exists an update in the set that brings the view back to the original state (thus canceling the effect of the previous update).

In order to translate a complete set, we use a mapping called a "translator." This mapping associates with each view update a unique database update, called a "translation." The constraint on a translation is to take the database to a state mapping onto the updated view. The constraint on the translator is to be morphism, that is, to associate with the composition of two view updates the composition of their translations.

We solve the following problem: Given a database view and a complete set U of view updates, find a translator of U. The solution is based on the concept of a view "complement." Together with the user-defined view, we define a complementary view such that the database could be computed from the view and its complement. We show that a view can have more than one complement and that the choice of a complement determines an update policy. Thus we fix a view complement, and we define the translation of a given view update in such a way that the complement remains invariant ("translation under constant complement"). The main result of the paper is the following: Given a complete set U of view updates,

(i) the translations of the updates in U under a constant complement define a translator of U;
(ii) to every translator of U, there corresponds a complement which remains invariant by the translations of the updates in U.

2. BASIC DEFINITIONS AND NOTATION

Let $A = \{A_1, A_2, \ldots, A_n\}$ be a set of names called *attributes*. With each attribute A_i we associate a set of values V_i (these sets of values need not be distinct). A mapping which assigns to each attribute A_i a value in V_i is called a *tuple* over A. A set of tuples is called a *relation* over A. It is not hard to see how the tuples of

a relation can be represented as lines in a table where each column is associated one-to-one with an attribute. The notation $R(A_1:V_1, A_2:V_2, \ldots, A_n:V_n)$ is used to represent a variable R whose values are relations over A. We shall call R a *relational variable.* When the attributes clearly imply the corresponding sets of values, the notation can be simplified. For example, we write PAY(EMPLOYEE, SALARY) instead of PAY(EMPLOYEE : NAMES, SALARY : INTEGERS). We shall use the notation $r.X$ to denote the restriction of a tuple r to a subset X of A. The projection of R over X, denoted by $R[X]$, is a relation over X defined as follows:

$$R[X] = \{r.X \mid r \in R\}.$$

Let R, W be two relational variables over attribute sets A and B, respectively. Let x denote tuples over $A \cup B$. The *join* of R and W, denoted by $R * W$, is a relation over $A \cup B$ defined as follows:

$$R * W = \{x \mid x.A \in R, x.B \in W\}.$$

A *database* is a set of relational variables together with a set of properties called *integrity constraints.* Functional dependencies are an important type of integrity constraint [1]. If R is a relational variable over A, then we may have two sets of attributes X, $Y \subset A$ such that for any two tuples t_1, t_2 of R, if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$. We then say that X functionally determines Y in R, denoted $X \rightarrow Y$ (if R is understood). A database *state,* denoted by s, is any assignment of values (i.e., relations) to the variables such that the integrity constraints are satisfied. The database *schema,* denoted by S, is the set of all database states. Figure 1 shows a database where E and M are the relational variables; C1, C2, and C3 are the integrity constraints; and s is the current database state; the database schema is the set $S = \{s \mid \text{C1, C2, C3}\}$. We shall refer to this example throughout the paper. Note that the current database state is sufficient to verify the integrity constraints. Such constraints are called *static.* Constraints requiring more than one database state for their verification are called *dynamic.* In this paper we restrict our attention to static constraints only.

A *view* is a database whose schema is derived from the schema of a given database. That is, in order to obtain a view of a given database with schema S,

(i) define a set of relational variables;
(ii) define a mapping f that associates with each database state $s \in S$ a view state $f(s)$.

The mapping f is the *view definition mapping.* The set $f(S) = \{f(s) \mid s \in S\}$ is the *view schema.* Figure 1 shows five different views of the same database. Views are normally defined using a relational data language. For example, using SEQUEL statements [7], VIEW #3 could be defined as follows:

```
DEFINE VIEW EM AS
SELECT EMPL, MGR
FROM E, M
WHERE E.DEPT = M.DEPT
```

The first two lines define a relational variable EM. The last three lines define the mapping f_3.

DATABASE

E(EMP, DEP)
M(DEP, MGR)
C1 : EMP → DEP
C2 : DEP ↔ MGR
C3 : E[DEP] = M[DEP]

CURRENT STATE s =

E

EMP	DEP
A	*l*
B	*l*
C	*m*

M

DEP	MGR
l	*X*
m	*Y*

VIEW #1
E(EMP, DEP)
$f_1(s)$ = E

$f_1(s)$ =

E

EMP	DEP
A	*l*
B	*l*
C	*m*

VIEW #2
M(DEP, MGR)
$f_2(s)$ = M

$f_2(s)$ =

M

DEP	MGR
l	*X*
m	*Y*

VIEW #3
EM(EMP, MGR)
$f_3(s)$ = (E * M)[EMP, MGR]

$f_3(s)$ =

EM

EMP	MGR
A	*X*
B	*X*
C	*Y*

VIEW #4
EDM(EMP, DEP, MGR)
$f_4(s)$ = E * M

$f_4(s)$ =

EDM

EMP	DEP	MGR
A	*l*	*X*
B	*l*	*X*
C	*m*	*Y*

VIEW #5
DC(DEP, #EMP)
$f_5(s) = \{x \mid x.\text{DEP} \in \text{E}, x.\#\text{EMP} = |A|\}$
where
$A = \{y \in \text{E} \mid y.\text{DEP} = x.\text{DEP}\}$

$f_5(s)$ =

DC

DEP	#EMP
l	2
m	1

Fig. 1. Five different views of a given database.

An *update* of a database with schema S can be seen as a mapping from S into S. This is so because an update is executed only if the result of the update operation satisfies the integrity constraints. Therefore, an update takes the current database state (whatever this state may be) to a new state in the schema. For example, in the database of Figure 1, the following statement is an update:

t : Fire employee B if he is not the only employee in his department.

If B does not appear in the current database state s, or if B is the only employee in his department, then t will not change s; that is, $t(s) = s$; otherwise, t will change s to a new state $t(s) \in S$ (this is the case in Figure 1). Therefore, we can think of the given database update as a mapping $t : S \rightarrow S$. In keeping with this definition of a database update, a *view update* can be seen as a mapping from the view schema into itself. However, the view schema is a derived schema, and this poses a consistency problem between the view and the underlying database. In Section 3, we discuss this and related problems in formal terms.

In our discussions throughout this paper,

S denotes the schema of a given database;
f denotes the definition mapping of a given view (we shall say, simply, "the view f");
s denotes the current database state;
U_1 denotes the set of all database updates;
U_f denotes the set of all view updates.

3. THE VIEW UPDATE PROBLEM

In order to state the view update problem, we must first answer the following questions:

Q1. Given a view update u, what are the constraints on the database update that translates 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 database update that translates it.

First, suppose the view f is updated by some $u \in U_f$. Then the database must also be updated by some $t \in U_1$, to reflect the view update operation. In order for t to be consistent with the view f, it must always take the database to a state that maps onto the updated view. That is, it must be such that $ft(s) = uf(s)$, $\forall s \in S$. But is every consistent t acceptable? Let us see an example. Refer to Figure 1 and consider the following update of VIEW #2:

u : In department n, replace manager Z by W.

Clearly, u does not change the current view state, in Figure 1. Consider now the following database update:

t : In department n, replace manager Z by W *and* hire one more employee in each existing department.

While t is consistent with f_2, it is not acceptable, as it changes the database although no changes are made in the view. Therefore, to answer Q1 above, we require that the database update be both consistent with the view and acceptable.

Definition 3.1. Given a view update $u \in U_f$, a database update $T_u \in U_1$ is called a *translation* of u iff

(i) $fT_u = uf$ (consistent), (concatenation denotes composition),
(ii) $\forall s \in S$, $uf(s) = f(s) \Rightarrow T_u(s) = s$ (acceptable). □

The next question to settle is defining the set of updates that the user of a view is allowed. User requirements impose two constraints on this set, namely,

(i) If the user is allowed two updates u and v, then he must also be allowed the composite update uv.
(ii) The user must have the means to cancel, if he wishes, the effect of every update that he is allowed on the view.

We state these requirements formally in the following definition of a complete set.

Definition 3.2. A set U of view updates is called *complete* iff

(i) $\forall u \in U, \forall v \in U, uv \in U$,
(ii) $\forall s \in S, \forall u \in U, \exists v \in U$ such that $vuf(s) = f(s)$. □

(Note that condition (ii) above does not imply that v is an inverse of u.) So to answer Q2 above, we want to translate complete sets of view updates.

Finally, in order to translate a complete set U, we need a mapping that associates with each view update in U a database update. The obvious constraints on T are

(i) With each view update $u \in U$, T must associate a translation of u.
(ii) Whether the user applies two updates from U one after the other, or their composite update, the result of the translation must be the same.

We state these requirements formally in the following definition of a translator.

Definition 3.3. Let $U \subset U_f$ be a complete set. A mapping $T : U \rightarrow U_1$ is called a *translator* iff

(i) $\forall u \in U$, $T(u)$ is a translation of u,
(ii) $\forall u \in U, \forall v \in U, T(uv) = T(u)T(v)$ (i.e., T is a morphism). □

To simplify notation, we shall write T_u instead of $T(u)$ throughout this paper. To answer Q3 above, we shall use translators to associate database updates to view updates in complete sets.

We are now ready to state the view update problem that we consider in this paper: given a complete set U of view updates, find a translator of U. We solve this problem as follows: with each $u \in U$ we associate a translation of u that leaves invariant "the information not visible within the view." We show that, if this is possible for all u in U, this association is a translator. Then we proceed to show that this is the *only* way of obtaining translators. That is, we show that, given any translator T of U, the translations T_u leave invariant "the information not visible within the view." The solution of the view update problem is based on a precise definition of the information not visible within the view. We study this problem in the next section.

4. THE VIEW COMPLEMENT

What is "the informatioin not visible within the view"? Let us see an example. Consider the database of Figure 1. It is clear that "the information not visible within VIEW #1" is the base relation E. It is less clear in the case of VIEW #2.

It looks as though either E or M qualifies as "the information not visible within the view." Therefore, we need a precise definition of this concept. First, let us introduce some basic definitions and notation.

We denote $M(S)$ a set of view definition mappings on S. We assume that $M(S)$ contains the identity mapping on S, denoted 1, and a constant mapping on S, denoted 0. An example is the set of all projection mappings on S. We do not specify here what the set $M(S)$ is, as the theory we present is independent of the type of mappings chosen. Only when we want to implement actual algorithms must we worry about this problem. What we do need here is a structure on the set $M(S)$, which we now define.

Definition 4.1. Let $f, g \in M(S)$. We say that f is greater than g or that f *determines* g, denoted $f \geq g$, iff

$$\forall s \in S, \quad \forall s' \in S, \quad f(s) = f(s') \Rightarrow g(s) = g(s'). \qquad \square$$

This definition can be interpreted as follows: $f \geq g$ iff whenever we know the state $f(s)$, we could compute the state $g(s)$. Following this interpretation, in the example of Figure 1, we conclude that $f_4 \geq f_3$, as EDM is sufficient to compute EM. Similarly, $f_4 \geq f_1$ and $f_4 \geq f_5$. On the other hand, f_2 and f_3 cannot be compared, as neither M or EM can be computed from the other. Definition 4.1 can also be interpreted using the partitions of S induced by f and g, that is, S/f and S/g, respectively. Recall that S/f is defined as follows: $\forall s \in S$, $\forall s' \in S$, s and s' are in the same member of S/f iff $f(s) = f(s')$. Also recall that S/f is called a refinement of S/g iff each member of S/f is contained in a member of S/g. The following theorem is an immediate consequence of Definition 4.1.

THEOREM 4.1. *$f \geq g$ iff S/f is a refinement of S/g.* $\square$

This theorem has some important implications. First, knowledge of the partitions S/f and S/g is sufficient to order f and g. In this respect, the mappings 1 and 0 are of particular interest. Each member of the partition $S/1$ consists of a single element of S; therefore, $\forall f \in M(S)$, $S/1$ refines S/f. It follows that $\forall f \in S/f$, $f \leq 1$. Similarly, the partition $S/0$ consists of just one member, namely, S itself. Therefore, $\forall f \in M(S)$, S/f refines $S/0$. It follows that $\forall f \in M(S)$, $0 \leq f$. The relation $\geq$ induces the following equivalence relation on the set $M(S)$.

Definition 4.2. Let $f, g \in M(S)$. We say that f and g are *equivalent*, denoted $f \equiv g$, iff $f \geq g$ and $g \geq f$. $\square$

As an immediate consequence of this definition we obtain $f \equiv g$ iff $S/f = S/g$. This implies that 1 is equivalent to every injective mapping on S, and that 0 is equivalent to every constant mapping on S. In Figure 1, we have $f_4 \equiv 1$, as EDM is sufficient to compute E and M, and conversely. Loosely speaking, VIEW #4 is equivalent to the database.

Let us refer again to Figure 1. It is clear that $f_2 \not\equiv 1$ and $f_3 \not\equiv 1$. That is, we cannot recompute the database using VIEW #2 alone or VIEW #3 alone. But it looks as though we can do so if we put the two views together and create a new view. So let us define a view definition mapping that "puts together" two given views.

Definition 4.3. Let $f, g \in M(S)$. The product of f and g, denoted $f \times g$, is defined by

$$f \times g(s) = (f(s), g(s)), \forall s \in S. \quad \square$$

As an example, the product of f_2 and f_5 (Figure 1), at the current database state s, is computed as follows:

$$f_2 \times f_5(s) = (f_2(s), f_5(s)) = (\mathrm{M}, \mathrm{DC}).$$

We assume that the set $M(S)$ is closed under product (i.e., $\forall f, g \in M(S)$, $f \times g \in M(S)$) so that we can compare products to other views. One consequence of Definition 4.3 is the following:

$$\forall f, g \in M(S),\ f \leq f \times g \quad \text{and} \quad g \leq f \times g.$$

That is, given a view f, the product $f \times g$ "adds" to f the information in g. Suppose now that $f \times g$ contains sufficient information for computing the database. Then g must contain the information not visible within the view f. This leads to the following definition of a view complement:

Definition 4.4. Let $f \in M(S)$. A view $g \in M(S)$ is called a *complement* of f iff $f \times g \equiv 1$. $\square$

(Note that f is a complement of g iff g is a complement of f.)

As an example, the views f_2 and f_3 (Figure 1) are complementary, whereas f_2 and f_5 are not. It follows from the previous definition that a view and its complement provide an isomorphic image of the schema S. That is, if f and g are complementary, then there is a one-to-one correspondence between database states s and the pairs $(f(s), g(s))$. Thus, for all practical purposes, s can be replaced by the pair $(f(s), g(s))$. We shall refer to this pair as a *representation* of s. Another interpretation of complements is provided by the following theorem, which is a consequence of Definition 4.4.

THEOREM 4.2. *Let $f, g \in M(S)$. g is a complement of f iff*

$$\forall s \in S, \forall s' \in S,\ f(s) = f(s') \Rightarrow g(s) \neq g(s'). \quad \square$$

What this theorem says is that g is a complement of f iff g separates or distinguishes between database states that map onto the same view state (under f). It follows that 1 is a complement of every view in $M(S)$. In particular, 1 is a complement of 0. Therefore, given any view f there always exists a complement of f. We shall make use of this fact in the following section.

Intuitively, if g is a complement of f, then any view larger than g is also a complement of f. Here is why: g adds sufficient information to f to recompute the database. Therefore, any view that adds to f at least as much information must be a complement of f. We state this formally in the following theorem. Its proof follows immediately from the previous theorem and *Definition* 4.1.

THEOREM 4.3. *Let g be a complement of f. Let $h \geq g$. Then h is a complement of f.* $\square$

As an example, f_3 is a complement of f_2 (Figure 1) and $f_4 \geq f_3$. Therefore f_4 is also a complement of f_2. There is, however, a basic difference between the two

complements f_3 and f_4: namely, it seems that there is no complement of f_2 which is smaller than f_3. In other words, f_3 is in some sense a minimal complement.

Definition 4.5. g is a *minimal* complement of f iff

(i) g is a complement of f,
(ii) if h is a complement of f and $h \leq g$, then $h \equiv g$. □

According to this definition, view f_4 (Figure 1) does not qualify as a minimal complement of f_2 (f_3 is a complement of f_2 such that $f_2 \leq f_3$ and $f_2 \not\equiv f_3$).

A minimal complement provides a formal description of the information not visible within the view. But it looks like this description is not unique. Refer again to Figure 1. Each of the views f_1, f_2 is a minimal complement of f_3. It turns out that only the trivial views 0 and 1 have unique (up to equivalence) minimal complements.

THEOREM 4.4. *Let $f \in M(S)$. f has a unique (up to equivalence) minimal complement iff $f \equiv 0$ or $f \equiv 1$.* □

PROOF

"If" part. If $f \equiv 0$, then f is a constant mapping. Therefore, g is a complement of f iff g is injective. It follows that g is a complement of f iff $g \equiv 1$. Therefore, 1 is the unique (up to equivalence) minimal complement of f.

If $f \equiv 1$, then f is an injective mapping. Therefore, $\forall g \in M(S)$, g is a complement of f. It follows that 0 is the unique (up to equivalence) minimal complement of f.

"Only if" part. It is enough to show that if

$$f \not\equiv 1, \tag{1}$$

$$f \not\equiv 0, \tag{2}$$

$$g \text{ is a minimal complement of } f, \tag{3}$$

then there exists complement h of f such that $h \not\equiv g$ and $g \not\leq h$. It follows from (1) that there exists $A \in S/f$ such that $|A| > 1$; that is,

$$\exists A \in S/f \quad \text{and} \quad a, a' \in A \quad \text{such that} \quad a \neq a'. \tag{4}$$

It follows from (2) that

$$\exists B \in S/f \quad \text{such that} \quad B \neq \emptyset \quad \text{and} \quad B \neq A. \tag{5}$$

Let $b \in B$. As g is a complement of f, it follows from (4) that

$$g(a) \neq g(a') \quad \text{(Theorem 4.2)}.$$

Therefore, either$g(a) \neq g(b)$ or $g(a') \neq g(b)$. Assume that

$$g(a) \neq g(b). \tag{6}$$

Define a view h on S such that

$$h(a) = h(b) = c, \qquad \text{where} \quad c \notin g(S), \tag{7}$$

$$h(s) = g(s), \forall s \in S \qquad \text{such that} \quad s \neq a \quad \text{and} \quad s \neq b. \tag{8}$$

As g is a complement of f, g distinguishes between all database states that map onto the same view state, under f (Theorem 4.2). It follows from (7) and (8) that h does also. Therefore h is also a complement of f (Theorem 4.2).

It follows from (6) and (7) that $S/g \neq S/h$. Therefore, $g \not\equiv h$. In order to show that $g \not\leq h$, it is enough to find two states s, $s' \in S$ such that $h(s) = h(s')$ and $g(s) \neq g(s')$ (Definition 4.1). This follows immediately from (6) and (7) if we take $s = a$ and $s' = b$. □

In conclusion, we have seen that

(i) given a view f, there always exists a complement of f;
(ii) a minimal complement of f describes the information not visible within f;
(iii) in general, a minimal complement is not unique.

In view of (iii), there is no unique way to describe "the information not visible within the view." Therefore, there is not much sense in asking that this information remain invariant during the translation process. Instead, we shall fix, in advance, a description of that part of database information that must remain invariant. We shall require, however, that this description be a view complement (not necessarily minimal). In this way, the view and its complement provide an isomorphic image of the schema. We shall use this fact in order to translate view updates into database updates.

5. TRANSLATION UNDER CONSTANT COMPLEMENT

The problem that we consider in this section can be stated as follows: given a complement g of the view f and a view update $u \in U_f$, define a translation of u that leaves the complement g invariant. The invariance of a complement corresponds to the "rectangle rule" in [4] and the "absence of side effects" in [5]. Let us remark right away that, intuitively, if g remains invariant by a translation of u, then any information contained in g must also remain invariant. The following theorem states that this is indeed the case.

THEOREM 5.1. *Let g be a complement of f. Let $u \in U_f$. Let t be a translation of u such that $gt = g$. Let h be a view such that $h \leq g$. Then $ht = h$.* □

PROOF. We have that $\forall s \in S$, $gt(s) = g(s)$. As $h \leq g$, it follows that $\forall s \in S$, $ht(s) = h(s)$, that is, $ht = h$. Q.E.D.

Let us now return to our original problem. As g is a complement of f, the current database state can be represented by the pair $(f(s), g(s))$. Updating $f(s)$ by u, while keeping $g(s)$ invariant, results in the pair $(uf(s), g(s))$. If this last pair lies in the image of S under $f \times g$, then it represents a unique database state s'. We shall associate s' to s to obtain the desired translation of u. This process is illustrated in Figure 2. First, however, we must make sure that the state s' exists.

Definition 5.1. Let $u \in U_f$. Let g be a complement of f. u is called *g-translatable* iff

$$\forall s \in S, \exists s' \in S \quad \text{so that} \quad f(s') = uf(s) \quad \textit{and} \quad g(s') = g(s). \qquad \square$$

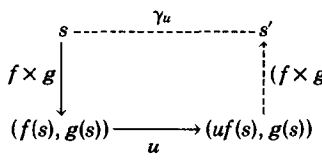

Figure 2

This definition simply states the existence condition for the state s' that leads to the desired translation of u.

THEOREM 5.2. *Let $u \in U_f$. Let g be a complement of f. Then u is g-translatable if $\forall s \in S, g(f^{-1}(f(s))) \subset g(f^{-1}(uf(s)))$.* □

PROOF. $\forall s \in S, s \in f^{-1}(f(s))$. Therefore, $\forall s \in S, g(s) \in g(f^{-1}(f(s)))$. It follows, from the hypothesis of the theorem, that $\forall s \in S, g(s) \in g(f^{-1}(uf(s)))$. Therefore $\forall s \in S, \exists s' \in f^{-1}(uf(s))$ such that $g(s') = g(s)$. Moreover, as $s' \in f^{-1}(uf(s))$, we have $f(s') = uf(s)$. It follows that u is g-translatable. Q.E.D.

Sets of g-translatable updates are closed under composition, as the following theorem shows.

THEOREM 5.3. *Let g be a complement of f. Let $u, v \in U_f$ be g-translatable. Then uv is g-translatable.* □

PROOF. As v is g-translatable, $\forall s \in S, \exists s' \in S$ such that (1) $f(s') = vf(s)$, and (2) $g(s') = g(s)$. As u is g-translatable, there exists $s'' \in S$ such that (3) $f(s'') = uf(s')$ and (4) $g(s'') = g(s')$. Therefore, $\forall s \in S, \exists s'' \in S$ such that

$$f(s'') = uvf(s) \qquad \text{(It follows from (1) and (3))}$$

$$g(s'') = g(s) \qquad \text{(It follows from (2) and (4)).}$$

Therefore uv is g-translatable. Q.E.D.

We show now that, for g-translatable updates, the association of s' to s (see previous diagram) produces the desirable translation of u. First, we give the following theorem, which is an immediate consequence of Definition 4.3.

THEOREM 5.4. *Let f, g, h, l be views in $M(S)$. Let a be a database update. Then*

(i) $(f \times g)a = fa \times ga$,
(ii) $f \times g = h \times l \Rightarrow f = h$ *and* $g = l$. □

($fa \times ga$ denotes the product of fa and ga.)

THEOREM 5.5. *Let g be a complement of f. Let $u \in U_f$ be a g-translatable update. Define $\gamma_u = (f \times g)^{-1}(uf \times g)$. Then*

(i) γ_u *is a translation of* u,
(ii) $g\gamma_u = g$. □

We call γ_u the *g-translation* of u.

PROOF. Note first that, as u is g-translatable, γ_u is a well-defined mapping from S into S; that is, γ_u is a database update. Also, noting that $(f \times g)(f \times g)^{-1}$ is the

identity mapping on S, we obtain

$$uf \times g = [(f \times g)(f \times g)^{-1}](uf \times g)$$
$$= (f \times g)[(f \times g)^{-1}(uf \times g)] = (f \times g)\gamma_u. \tag{9}$$

(i) To show that γ_u is a translation of u, we must show that it is consistent with the view and acceptable (Definition 3.1).

Consistent. It follows from (9) that $uf \times g = (f \times g)\gamma_u$. Then Theorem 5.3 implies that $uf = f\gamma_u$. Q.E.D.

Acceptable. Let $s \in S$. Suppose $uf(s) = f(s)$. We must show that $\gamma_u(s) = s$. We have

$$\gamma_u(s) = (f \times g)^{-1}(uf \times g)(s)$$
$$= (f \times g)^{-1}(uf(s), g(s))$$
$$= (f \times g)^{-1}(f(s), g(s)) \qquad \text{(because of our hypothesis)}$$
$$= (f \times g)^{-1}(f \times g)(s)$$
$$= s. \qquad \text{Q.E.D.}$$

(ii) It follows from (9) that $uf \times g = (f \times g)\gamma_u$. Then Theorem 5.4 implies that $g = g\gamma_u$. Q.E.D.

Let us now see a complete example of a translation. Refer again to Figure 1 and consider the following update on VIEW #3:

u: Replace employee A by employee F.

Suppose that we declare f_2 as the complement of f_3 (that must remain invariant during translation). Then the update u is f_2-translatable. In terms of f_2 and u, the translation γ_u is defined as follows:

E := (M * u(EM))[EMP, DEP]
M := M

(The symbol := denotes assignment of a value.)

Figure 3 shows the details of the translation process.

It should be noted that this example is only meant to explain the translation process. It must not be interpreted as a method for the actual computation of the updated database.

We conclude this section by an important property of g-translations: namely, we show that uniqueness of g-translations is *implied* by the invariance of the complement. Therefore, it is not necessary to state it as a constraint on the translation process.

THEOREM 5.6. *Let g be a complement of f. Let $u \in U_f$. Let t_u be a translation of u such that $gt_u = g$. Then*

(i) u is g-translatable,
(ii) $t_u = (f \times g)^{-1}(uf \times g)$ (i.e., t_u is the g-translation of u). □

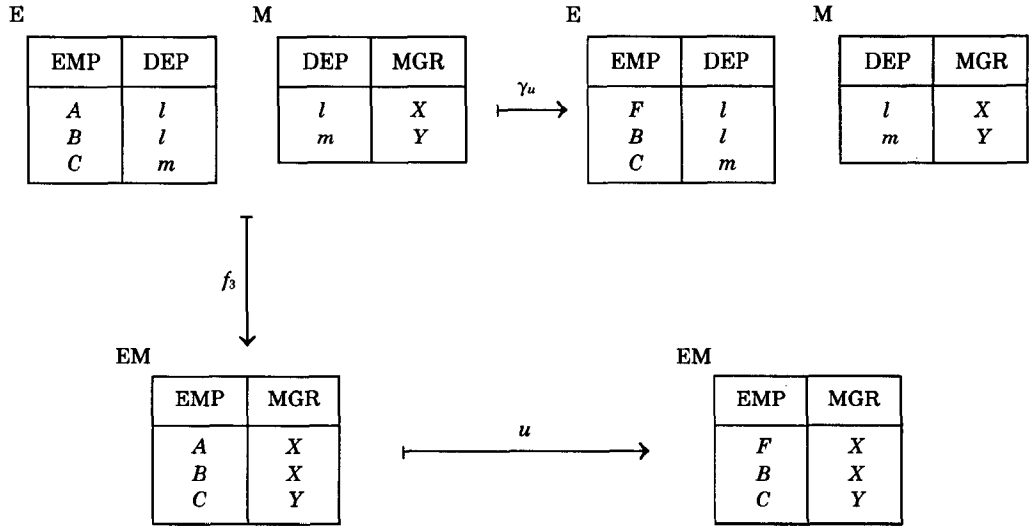

Fig. 3. Details of the translation process.

PROOF

(i) Let $s \in S$. Let $t_u(s) = s'$. It follows that

$$f(s') = ft_u(s) = uf(s) \qquad \text{(because } t_u \text{ is a translation of } u\text{)}$$

$$g(s') = gt_u(s) = g(s).$$

It follows from Definition 5.1 that u is g-translatable. Q.E.D.

(ii) As g is a complement of f, $f \times g$ is injective. As u is g-translatable, $uf \times g(S) \subset f \times g(S)$. If we restrict the codomain of $f \times g$ to the set $f \times g(S)$, then $f \times g$ becomes bijective. On the other hand, we have

$$(f \times g)t_u = ft_u \times gt_u \qquad \text{(from Theorem 5.4)}$$

$$= uf \times g \qquad \text{(because } t_u \text{ is a translation and } gt_u = g\text{)}.$$

Therefore, $(f \times g)t_u = uf \times g$ and, as $f \times g$ is bijective, we obtain $t_u = (f \times g)^{-1}(uf \times g)$. Q.E.D.

There is an important consequence of this theorm: To every triple (f, g, u) such that g is a complement of f and u is g-translatable, there corresponds one and only one translation of u that leaves g invariant. This is precisely the g-translation γ_u.

In [5] it is argued that "an update on a view f is translatable if there is a *unique* database update producing the desired update of f such that . . . ." Then, later, the authors remark that the uniqueness criterion is controversial: "we do not feel comfortable with this choice but could not find more attractive alternatives . . . admittedly, our correctness criteria have weaknesses. However, these weaknesses are due to gaps in our understanding of update semantics in the relational model." Indeed, it seems that the uniqueness criterion is the best choice when the only

information available is the view definition. However, as the authors observe, "the view definition mapping ... is often not sufficient to express the semantics of the relationship between update operations on the view and on the [original] schema."

Our conclusion here is that the extra information that we need in order to express these semantics is precisely a view complement.

6. THE CHOICE OF A COMPLEMENT

We have seen that a given view update may or may not be translatable depending on the choice of a view complement. Let us see an example. Consider the following updates on VIEW #3 (Figure 1):

u: Replace employee *A* by employee *F*.
v: Permute the managers.

If we declare f_2 as the complement of f_3, then u is f_2-translatable, whereas v is not. In other words, if we decide that the association of departments and managers (in the database) must not change while updating VIEW #3, then only u can be translated (with respect to this "policy"). Alternately, if we declare f_1 as the complement of f_3, then v is f_1-translatable, whereas u is not. In other words, if we decide that the association of employees and departments (in the database) must not change while updating VIEW #3, then only v can be translated (with respect to this new "policy"). Therefore, a view complement corresponds to a view update policy. The existence of more than one view complement simply means that there are more than one view update policies. The choice of a policy depends, of course, on the specific application. But once this choice is made we must check whether a given view update is translatable with respect to this policy (g-translatability). And if it is, we must compute its translation (g-translation).

Let us see one more example. Consider a database, a user-defined view f, and three different update policies (i.e., view complements) g_1, g_2, g_3, as follows:

DATABASE:	R(PRODUCT, COST, SALEPRICE, PROFIT, PROFITRATE)
	C1: PRODUCT $\rightarrow$ COST, SALEPRICE, PROFIT, PROFITRATE
	C2: $\forall x \in R$, x.COST ≥ 0
	x.SALEPRICE $\geq x$.COST
	x.PROFIT = x.SALEPRICE − x.COST
	x.PROFITRATE = x.PROFIT/x.COST
VIEW:	PC(PRODUCT, COST)
	$f(s) = R$[PRODUCT, COST]
POLICY #1:	PS(PRODUCT, SALEPRICE)
	$g_1(s) = R$[PRODUCT, SALEPRICE]
POLICY #2:	PP(PRODUCT, PROFIT)
	$g_2(s) = R$[PRODUCT, PROFIT]
POLICY #3:	PR(PRODUCT, PROFITRATE)
	$g_3(s) = R$[PRODUCT, PROFITRATE]

Suppose that, in the context of view f above, we want to perform the following update:

u: Decrease the cost of all product by 10 percent.

Note that g_i is a complement of f and that u is g_i-translatable, $i = 1, 2, 3$. In all three cases, the cost will decrease by 10 percent (at the database level) as

required. But the changes made in the remaining part of the database depend on the policy (i.e., the view complement) chosen. Thus, along with the 10 percent decrease in cost, the following changes are made in the database:

POLICY #1. The sale price remains constant, but profit and profit rate increase.

POLICY #2. The profit remains constant, the sale price decreases, and the profit rate increases.

POLICY #3. The profit rate remains constant, but profit and sale price decrease.

The choice of a complement g for a given view f determines the set U of view updates that are g-translatable. Of course, these are the only updates that a user of the view is allowed to perform. Intuitively, we would expect the set U to become larger as the complement g becomes smaller. The following theorem says that this is indeed the case.

THEOREM 6.1. *Let g, h be complements of f such that $h \leq g$. Let $u \in U_f$ be g-translatable. Then*

(i) u is h-translatable,
(ii) the h-translation of u is equal to the g-translation of u. □

PROOF

(i) Let $s \in S$. As u is g-translatable, there exists $s' \in S$ such that

$$f(s') = uf(s), \tag{10}$$

$$g(s') = g(s). \tag{11}$$

As $h \leq g$, it follows from (11) that

$$h(s') = h(s). \tag{12}$$

It follows from (10) and (12) that u is h-translatable. Q.E.D.

(ii) Let γ_u be the g-translation of u. Then, $\forall s \in S$, $g\gamma_u(s) = g(s)$. As $h \leq g$, it follows that $\forall s \in S$, $h\gamma_u(s) = h(s)$. Therefore, $h\gamma_u = h$. It follows from Theorem 5.6 that γ_u is the h-translation of u. Q.E.D.

In view of this theorem, the set U of g-translatable updates is maximal when the complement g is minimal. We have seen that minimal complements are not necessarily unique. Therefore, there are as many maximal U's as there are minimal complements of the view.

Note that to every pair (f, g), where f and g are complementary views, there corresponds a set $U \subset U_f$ of g-translatable updates. There are two extreme cases, where $U = \emptyset$ and $U = U_f$. In the first case, f and g are such that no view update is g-translatable. As an example, consider VIEW #5, in Figure 1. f_4 is a complement of f_5, but no update of f_5 is f_4-translatable. Note, however, that the converse is not true. The following update of f_4 is f_5-translatable:

u: Permute the managers of departments l and m.

In the second case, where $U = U_f$, every update of f is g-translatable. Therefore, as far as updating goes, f is "independent" of g. The problem of finding independent views is extremely important in database theory. A complete treatment of this problem is presented in [3].

7. A UNIVERSAL PROPERTY

In this section we present the main result of this paper, which can be stated as follows. Given a complete set $U \subset U_f$:

(i) For every complement g of f such that: u is g-translatable $\forall u \in U$, the mapping $T: U \to U_1$ defined by: $\forall u \in U$, $T_u = \gamma_u$, is a translator of U.
(ii) To every translator T of U there corresponds a complement g of f such that: $\forall u \in U$, u is g-translatable and $\gamma_u = T_u$.

THEOREM 7.1. *Let $U \subset U_f$ be a complete set. Let g be a complement of f such that $\forall u \in U$, u is g-translatable. Then the mapping $T: U \to U_1$ defined by $T_u = \gamma_u$ is a translator of U.* □

PROOF. As γ_u is a translation, it follows from Definition 3.2 that it is enough to show that

(i) $\forall s \in S$, $uf(s) = f(s) \Rightarrow \gamma_u(s) = s$,
(ii) $\forall u \in U$, $\forall v \in U$, $\gamma_{uv} = \gamma_u \gamma_v$.

(i) We have

$$\begin{aligned}
\gamma_u(s) &= (f \times g)^{-1}(uf \times g)(s) && \text{(from Theorem 5.4)} \\
&= (f \times g)^{-1}(uf(s), g(s)) \\
&= (f \times g)^{-1}(f(s), g(s)) && \text{(from our hypothesis)} \\
&= (f \times g)^{-1}(f \times g)(s) \\
&= s. && \text{Q.E.D.}
\end{aligned}$$

(ii) Let $s \in S$. Let $\gamma_v(s) = s''$. Then

$$\begin{aligned}
s'' &= \gamma_v(s) \\
&= (f \times g)^{-1}(vf \times g)(s) \\
&= (f \times g)^{-1}(vf(s), g(s)).
\end{aligned}$$

It follows that (1) $f(s'') = vf(s)$, and (2) $g(s'') = g(s)$. Now,

$$\begin{aligned}
\gamma_u \gamma_v(s) &= \gamma_u(s'') && \text{(because } \gamma_v(s) = s'') \\
&= (f \times g)^{-1}(uf \times g)(s'') \\
&= (f \times g)^{-1}(uf(s''), g(s'')) \\
&= (f \times g)^{-1}(uvf(s), g(s)) && \text{(from (1) and (2))} \\
&= (f \times g)^{-1}(uvf \times g)(s) \\
&= \gamma_{uv}. && \text{Q.E.D.}
\end{aligned}$$

To prove the converse of this theorem, we need the following result.

THEOREM 7.2. *Let $U \subset U_f$ be a complete set. Let T be a translator of U. The following relation on S is an equivalence relation:*

$$\forall s \in S, \forall s' \in S,\ s \equiv s' \quad \text{iff} \quad \exists u \in U \ \text{ such that } \ s = T_u(s'). \ \square$$

PROOF. We show that $\equiv$ is reflexive, symmetric, and transitive.

Reflexivity. Let $s \in A$, for some $A \in S/f$. Take any $u \in U$. If $T_u(s) \in A$, then $uf(s) = f(s)$, and this implies that $s = T_u(s)$, that is, $s \equiv s$ (see Definition 3.2). If $T_u(s) \notin A$, then $uf(s) \neq f(s)$ and, as U is complete, there exists $v \in U$ such that $vuf(s) = f(s)$. This implies that $s = T_{vu}(s)$ and, as $vu \in U$, it follows that $s \equiv s$. Q.E.D.

Symmetry. Let $s, s' \in S$ such that $s \neq s'$ (if $s = s'$, we can use reflexivity). suppose that $s \equiv s'$. Then there exists $u \in U$ such that $s = T_u(s')$. Then, supposing that $s \in A$, for some $A \in S/f$, we obtain $T_u(s') \in A$. This implies that $s' \notin A$ (for, if $s' \in A$, then $s' = T_u(s') = s$, a contradiction). It follows that $fT_u(s') \neq f(s')$ and, as $fT_u = uf$, we obtain $uf(s') \neq f(s')$. Now, as U is complete, there exists $v \in U$ such that $vuf(s') = f(s')$. It follows that $s' = T_{vu}(s') = T_v(T_u(s')) = T_v(s)$. Therefore $s' \equiv s$. Q.E.D.

Transitivity. Suppose $s \equiv s'$ and $s' \equiv s''$. Then there exist $u, v \in U$ such that $s = T_u(s')$ and $s' = T_v(s'')$. It follows that $s = T_u(s') = T_u(T_v(s'')) = T_{uv}(s'')$. Therefore $s \equiv s''$. Q.E.D.

In order to show the converse of Theorem 7.1, we shall make use of the equivalence relation $\equiv$ of Theorem 7.2. We shall denote by $\bar{s}$ the equivalence class of s and by $S/\equiv$ the set of all equivalence classes of S.

THEOREM 7.3. *Let $U \subset U_f$ be a complete set. Let T be a translator of U. Define the mapping $g: S \to S/\equiv$ such that $\forall s \in S$, $g(s) = \bar{s}$. Then,*

(i) g is a complement of f;
(ii) $\forall u \in U$, u is g-translatable;
(iii) $\forall u \in U$, $T_u = (f \times g)^{-1}(uf \times g)$. $\square$

PROOF. (i) We must show that $f \times g$ is injective. That is, we must show that $\forall s, s' \in S, s \neq s' \Rightarrow f \times g(s) \neq f \times g(s')$, or, equivalently that $\forall s, s' \in S, s \neq s'$ *and* $g(s) = g(s') \Rightarrow f(s) \neq f(s')$. Supposing $g(s) = g(s')$, we obtain $\bar{s} = \bar{s'}$, that is, $s \equiv s'$. It follows that there exists $u \in U$ such that (1) $s = T_u(s')$. Now, if $f(s) = f(s')$, then (1) implies that $fT_u(s') = f(s')$, that is, $uf(s') = f(s')$. As T is a translator of u, we obtain $T_u(s') = s'$. It follows from (1) that $s = s'$, a contradiction. Therefore $f(s) \neq f(s')$. Q.E.D.

(ii) and (iii) follow directly from Theorem 5.5 if we show that $\forall u \in U$, $gT_u = g$. And in order to show this it is enough to show that $\forall s \in S$, $\forall u \in U$, $s \equiv T_u(s)$. Take any $s \in S$ and $u \in U$ and set $T_u(s) = s'$. It follows that $fT_u(s) = f(s')$, that is, (1) $uf(s) = f(s')$.

Suppose first that $f(s) = f(s')$. Then (1) implies $uf(s) = f(s)$. As T is a translator, we obtain $T_u(s) = s$, that is, $s \equiv T_u(s)$. Suppose next that $f(s) \neq f(s')$. Then (1) implies $uf(s) \neq f(s)$. As U is complete, there exists $v \in U$ such that $vuf(s) = f(s)$. As T is a translator, we obtain $T_{vu}(s) = s$ or $T_v(T_u(s)) = s$, that is, $s \equiv T_u(s)$. Q.E.D.

In conclusion, translation under constant complement is the only method of translation, in the sense that

(i) it provides translators for complete sets;
(ii) any translator of a complete set translates under some constant complement.

8. CONCLUSION

We have presented a formal treatment of the view update problem. The statement of the problem relies on three basic concepts;

(i) the complete set of view updates, which reflects user requirements;
(ii) the translation of a view update, which guarantees consistency between the view and the database;
(iii) the translator, which associates translations to view updates of complete sets.

The solution to the problem of finding translators is based on the key concept of a view complement. Roughly speaking, a view complement is database information that, along with the view, is sufficient to recompute the whole database. We have seen that, in general, view complements are not unique and that the choice of a complement can be interpreted as the definition of an update policy.

The method of solution can be summarized as follows:

(1) choose a complement g of the given view f;
(2) check whether the view updates of the given complete set U leave g invariant;
(3) associate with each $u \in U$, the translation $T_u = (f \times g)^{-1}(uf \times g)$.

We have shown that this method leads to a translator T of the complete set U. We have also shown that this is the only method of solution. Steps (1) and (2) of the solution depend on the given database schema S and the view definition mapping f. That is, they depend on the type of database integrity constraints and the operators allowed for view definition. Therefore, computational algorithms (if they exist) implementing steps (1) and (2) must be sought in specific problems: for example, schemata defined by functional dependencies and views derived by projections. For step (3), the most likely implementation will be by a program.

REFERENCES

(Note. Reference [6] is not cited in the text.)

1. ARMSTRONG, W.W. Dependency structures of data base relationships. *Information Processing 74*, North-Holland, Amsterdam, 1974, pp. 580–583.
2. ASTRAHAN, M.M., ET AL. System R: Relational approach to database management. *ACM Trans. Database Syst. 1*, 2 (June 1976), 97–137.

3. BANCILHON, F.M., AND SPYRATOS, N. Independent views of relational data bases. To appear.
4. CHAMBERLIN, D.D., ET AL. Views, authorization and locking in a relational data base system. In *Proc. 1975 Nat. Computer Conf.*, AFIPS Press, Arlington, Va.
5. DAYAL, U., AND BERNSTEIN, P.A. On the updatability of relational views. In *Proc. 4th VLDN Conf.*, West Berlin, Sept. 1978.
6. SEVCIK, K.C.., AND FURTADO, A.L. Complete and compatible sets of update operations. In *Proc. ICMOD 78 Conf.*, Milano, Italy, June 1978, pp. 247–260.
7. STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept. 1976), 189–222.

Received October 1978; revised April 1980; accepted January 1981