Data base mappings-Part 2 : application

François Bancilhon, N. Spyratos

INRIA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N°63

DATA BASE MAPPINGS
PART II : APPLICATIONS

François BANCILHON
Nicolas SPYRATOS

Avril 1981

DATA BASE MAPPINGS
PART II : APPLICATIONS

François Bancilhon - Nicolas Spyratos
INRIA
Domaine de Voluceau
B.P. 105
78153 Le Chesnay - France

Abstract

In this second part of the paper we apply our theory of data base mappings to two important areas of data base systems : data base decomposition and view updating. We show how our general results reduce to well known results in the special case of projections and functional dependencies. We also discuss some parallels between our theory of data base mappings, on the one hand, and the theory of relations and probability theory, on the other hand. Technical results concerning data base mappings are presented in a separate INRIA research report n° 62 under the title "Data base mappings, Part I : Theory".

Résumé

Nous appliquons la théorie des opérateurs relationnels définie précédemment dans un autre rapport INRIA n° 62 ("Data base mappings Part I : Theory") à deux données importantes des Bases de Données : la décomposition des bases de données et la mise à jour dans les vues. Nous montrons que les résultats connus sur les projections et les dépendences fonctionnelles sont des applications de nos résultats généraux. Nous présentons ainsi les liens entre notre théorie et les théories des relations et des probabilités.

INTRODUCTION

In this second part of the paper we apply the theory presented in the first part to data base decomposition and view updating.

The process of data base decomposition has been studied extensively in connection with data dependencies[16] and normal forms[17]. Also, decompositions into independent components have been proposed[7,24,25]. The common assumption is that the data base consists of one relation which is decomposed by projections, and the original relation is then reconstructed from the components using join. We study here decomposition from a general point of view. That is, we do not limit ourselves to the case of projection and join. In doing so, we have in mind applications where projection and join alone are not sufficient. Consider for example a distributed data base where distribution onto the various sites is done by decomposition. The usual operations by which the local data bases are obtained are projection and selection. This implies, among other things, that join alone is not sufficient in order to reconstruct the original data base.

The problem of translating view updates into data base updates has received considerable attention in the litterature[14,26,27]. A method for constructing translators of view updates has been recently proposed[4], based on the concept of a "view complement". That is, together with the user defined view, a complementary view is defined such that the data base can be computed from the view and its complement. Those view updates that leave the complement invariant can be translated in a unique way. Two questions that remain open in this context are the following

1 - Given a view update and a view complement decide whether the update is translatable

2 - Given a view and a view complement determine the set of all view updates that leave the complement invariant

We answer these questions applying the theory of data base mappings presented in the first part of this paper.

The notation introduced in the first part of the paper is used here without further explanation.

DATA BASE DECOMPOSITION

We shall define here a data base decomposition as a pair of data base mappings. These mappings specify the components into which the data base is decomposed. We shall restrict ourselves to decompositions into two components. Let us recall that, given a data base schema $<D|C>$, the symbol MAP stands for the set of all data base mappings on the set $S<D|C>$. That is, the set of all mappings f of the form

$$f : S<D|C> \rightarrow S<D'|\lambda>$$

Given a data base mapping

$$fi : S<D|C> \rightarrow S<Di|\lambda> \quad , \quad i=1,2,...$$

we shall adopt the following notational conventions

1 - S stands for $S<D|C>$

2 - Si stands for fi(S)

3 - Ci stands for any predicate on $S<Di|\lambda>$ such that

$$Ci(Di_t) = true \iff Di_t \in S_i$$

Given two mappings f1 and f2,

4 - D12 stands for the set $D1 \cup D2$

5 - S12 stands for $(f1 \times f2)(S)$

6 - C12 stands for any predicate on $S<D12|\lambda>$ such that

$$C12(D12_t) = true \iff D12_t \in S12$$

Definition 1 - Given the data base schema $<D|C>$, a decomposition is any pair of mappings (f1,f2) such that f1, f2 $\in$ MAP.
The result of the decomposition is the data base schema $<D12|C12>$ □
(We shall always assume that $D1 \cap D2 = \emptyset$)

Example 1

Relation schemas : R(EMPL,DEPT,MGR), R1(EMPL,DEPT), R2(DEPT,MGR)

Integrity constraints : c1 : EMPL → DEPT, c2 : DEPT → MGR,
c3 : R1[DEPT] = R2[DEPT], C = c1 ∧ c2, C12 = C ∧ c3

Data base variables : D = {R}, D1 = {R1}, D2 = {R2},
D12 = D1 ∪ D2 = {R1,R2}

Mappings : f1 = R[EMPL,DEPT], f2 = R[DEPT,MGR]

The pair (f1,f2) is a decomposition of $S<D|C>$ and its result is the data base schema $<D12|C12>$ □

Example 2

Relation schemas : R(NAME,AGE,ADDR), YOUNG(NAME,AGE,ADDR),
OLD(NAME,AGE,ADDR)

Integrity constraints : C : NAME → AGE,ADDR in R,
c1 : NAME → AGE,ADDR in YOUNG, c2 : NAME → AGE,ADDR in OLD,
c = YOUNG[NAME] = OLD[NAME], C12 = c1 ∧ c2 ∧ c

Data base variables : D = {R}, D1 = {R1}, D2 = {R2},
D12 = D1 ∪ D2 = {R1,R2}

Mappings : f1 = (R|AGE < 30), f2 = (R|AGE ≥ 30)

The part (f1,f2) is a decomposition of $S<D|C>$ and its result is the data base schema $<D12|C12>$ □

Let us now see what constitutes a "good" decomposition. We shall impose that such a decomposition must allow us to

1 - reconstruct the data base from its components

2 - operate separately on the components

Definition 2 - The decomposition (f1,f2) is called lossless iff f1 and f2 are complementary □

That is, in a lossless decomposition there exists a 1-1 mapping $(f1 \quad f1)^{-1}$ that reconstructs the data base without loss of information. Lossless decompositions have been studied extensively in the literature[1], in the context of projections. The following theorems are two basic results from this area.

Theorem 1[24]- Let $<\{R(XYZ)\}|F>$ be a data base schema, where F is a full family of functional dependencies on XYZ. The decomposition (R[XY], R[XZ]) is IP iff $(X \rightarrow Y) \in F$ or $(X \rightarrow Z) \in F$ □

Theorem 2[18]- Let $<\{R(XYZ)\}|C>$ be a data base schema. If $X \twoheadrightarrow Y$ then the decomposition (R[XY],R[XZ]) is lossless □

Let us emphasize that the converse of Theorem 2 is true only if we require that reconstruction of the data base be done by join. Let us see an example.

Example 3

Relation schemas : R(COURSE,STUD,GRADE,AVG), R1(COURSE,STUD,GRADE), R2(COURSE,AVG)

Integrity constraints : c1 : COURSE,STUD → GRADE in R
c2 : STUD → AVG in R, C = c1 ∧ c2

Data base variables : D = {R}, D1 = {R1}, D2 = {R2}, D12 = D1 ∪ D2 = {R1,R2}

Mappings : f1 = R[COURSE,STUD,GRADE], f2 = R[COURSE,AVG]

The decomposition (f1,f2) is lossles but COURSE ↠ AVG is not true in R. □

Let us now turn to the second requirement for a "good" decomposition. Namely, that we should be able to operate separately on the components. This implies that we should be able to verify the consistency of the decomposed data base, by verifying separately the consistency of each component. In other words, we should have : C12 = c1 ∧ c2.

Definition 3 - The decomposition (f1,f2) is called independent iff $C12 = c1 \wedge c2$ □

The interest in independent decomposition is obvious, especially in the case of a distributed data base, where the components are on different sites. Each site checks the local data base for consistency, and local consistency implies then global consistency. The name "independent" is further justified in view of the following theorem whose proof is trivial.

Theorem 2 - Given a decomposition (f1,f2), the following statements are equivalent

(1) (f1,f2) is independent

(2) $S<D12|C12> = S<D1|C1> \times S<D2|C2>$

(3) f1 and f2 are independent □

It follows from this theorem that it is not possible to give meaningful examples of independent decompositions using projections. The problem lies in the fact that : $f1 \sim f2 \Rightarrow f1 \wedge f2 \equiv 0_S$, that is, independent projections means projections on disjoint sets of attributes (and the universal relation assumption does not allow this !)

As we have seen earlier (examples 1 and 2), a lossless decomposition is not necessarily independent (and vice-versa). In general, a decomposition is neither lossless nor independent. Lack of independence, in particular, implies that before we apply a legal update on one of the two components we must take into account the effect on the other component. The question that we study next is the following. What are the updates that can be performed on one component whitout checking their effect on the other. In order to state this question formally, let us introduce some notation. Consider the state of the decomposed data base at time t

$$(D1_t, D2_t) \in S<D12|C12>$$

An update u on $D1_t$ is admissible, with respect to $D2_t$, iff

$$(u(D1_t), D2_t) \in S<D12|C12>$$

We define now the set of updates that we can apply on $D1_t$ without affecting $D2_t$

$$ADM(D1_t|D2_t) = \{u|u \text{ admissible with respect to } D2_t\}$$

This defines a function that associates to each pair $(D1_t, D2_t)$ of the decomposed data base, the corresponding set $ADM(D1_t|D2_t)$.

We shall denote this function by ADM(D1|D2). Let us see the intuitive meaning of this function in the case of a distributed data base. That is, D1 and D2 are at different sites, and a user updating D1 at time t knows only $D1_t$ and not $D2_t$. Two things can happen

1 - The function ADM(D1|D2) depends on D2 and, therefore, it is not known by the user (since he does not know $D2_t$). In this case he can not know what updates he can apply on $D1_t$ without consulting $D2_t$.

2 - The function ADM(D1|D2) does not depend on D2, that is, it is just a function of D1. In this case a user looking at $D1_t$, knows the set of all updates that he can perform on it.

So, in the second case the two components D1 and D2 can be updated independently.

Definition 4 - The decomposition (f1,f2) is called weakly independent iff the function ADM(D1|D2) is independent of D2 and the function ADM(D2|D1) is independent of D1 □

Having defined weak independence, the problem is to

1 - determine when the function ADM(D1|D2) is independent of D2

2 - "compute" the set $ADM(D1_t|D2_t)$, in this case.

The following theorem answers the first question

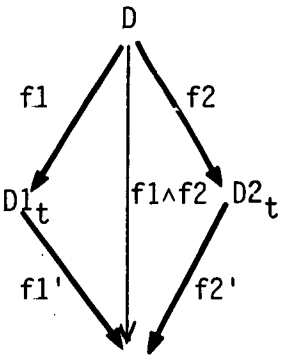
Theorem 3 - Given a decomposition (f1,f2), the following statements are equivalent

(1) ADM(D1|D2) is independent of D2

(2) ADM(D2|D1) is independent of D1

(3) f1 and f2 are weakly independent □

Proof : (3) => (1)



The fact that

$$f1 \wedge f2 \leq f1 \text{ and } f1 \wedge f2 \leq f2$$

implies that

$\exists$ f1', f2' such that

$$f1 \wedge f2 = f1'f1 = f2'f2$$

On the other hand, as f1 and f2 are weakly independent, that is, $f1 \sim f2 \bmod f1 \wedge f2$, we have that

$$\forall D1_t, \forall D2_t, \grave{} \; f1'(D1_t) = f2'(D2_t) \Rightarrow (D1_t, D2_t) \in S12$$

It follows that

$$\begin{aligned} ADM(D1_t|D2_t) &= \{u|(u(D1_t), D2_t) \in S12\} \\ &= \{u|f1'(u(D1_t)) = f2'(D2_t)\} \\ &= \{u|f1'(u(D1_t)) = f1'(D1_t)\} \end{aligned}$$

Therefore, the set $ADM(D1_t|D2_t)$ is independent of $D2_t$, Q.E.D.

(1) => (3)

As the function ADM(D1|D2) is independent of D2, we can consider it as a function on S1. This function induces the following partition on S1

$$\{ADM(D1_t|D2_t)\,|\,D1_t \in S1\}$$

Similarly, the function ADM(D2|D1) induces the following partition on S2

$$\{ADM(D2_t|D1_t)\,|\,D2_t \in S2\}$$

We can set the members of these two partitions into one-to-one correspondence as follows :

$ADM(D1_t|D2_t)$ is associated to $ADM(D2_t|D1_t)$ iff $(D1_t,D2_t) \in S12$

(It is easy to check that if this association is not a bijection then ADM(D1|D2) depends on D2).

Therefore, we can define two mappings f1' and f2' such that

$$f1'(D1_t) = f2'(D2_t) \text{ iff } ADM(D1_t|D2_t) \text{ and } ADM(D2_t|D1_t)$$

are associated.

It follows (from the definition of ADM) that

$$(D1_t,D2_t) \in S12 \iff f1'(D1_t) = f2'(D2_t)$$

Define now the mapping $h = f1'f1 = f2'f2$. We have

$f1(B) \times f2(B) = (f1 \times f2)(B) \quad \forall\, B \in S/h$, that is,

$$f1 \sim f2 \bmod h \qquad (a)$$

It follows from the definition of h that

$$h \leq f1 \text{ and } h \leq f2 \qquad (b)$$

It follows from (a) and Theorem 9 (PART I of this paper) that

$$h \geq f1 \wedge f2 \qquad (c)$$

It follows from (b) and (c) that $h \equiv f1 \wedge f2$, and (a) implies that f1 and f2 are weakly independent Q.E.D.

Thus, we have determined when the two components of a decomposition can be updated independently. What remains now is to "compute" the set of updates that we can perform on one component without consulting the value of the other.

Theorem 4 - Let (f1,f2) be a weakly independent decomposition. Let f1', f2' be the mappings such that

$$f1'f1 = f2'f2 = f1 \wedge f2$$

Then, u is admissible iff $f1'u = f1'$ □

Proof : We have that

$$\begin{aligned} u \text{ admissible} &\iff (u(D1_t), D2_t) \in S12 \\ &\iff f1'(u(D1_t)) = f2'(D2_t) \\ &\iff f1'(u(D1_t)) = f1'(D1_t) \\ &\iff f1'u(D1_t) = f1'(D1_t) \end{aligned}$$

Q.E.D.

It follows from this theorem that admissible are those updates that leave invariant the part $f1 \wedge f2$. Looking back at Example 1, we have $f1 \wedge f2 = R[DEPT]$. Therefore, admissible are all updates that leave invariant the list of departments. Let us see now an example of decomposition using selection.

Example 4

Let us consider a data base describing a set of students each following at least one of two courses : Math or Latin. We would like to consider these students in three groups

1 - Science students : Those who take Math

2 - Classics students : Those who take Latin

3 - General students : Those who take Math and Latin

Relation schemas : R(STUD,COURSE,GRADE), SC(STUD,COURSE,GRADE), CL(STUD,COURSE,GRADE), GEN(STUD,COURSE,GRADE)

Integrity constraints : c1 : STUD,COURSE → GRADE
c2 :((R|(COURSE=Math) ∨ (COURSE=Latin))[STUD])= R[STUD]
C = c1 ∧ c2

Data base variables : D = {R}, D1 = {SC}, D2 = {CL},
D3 = {GEN}

Mappings : f1 = (R|(s,Math) ∈ R[STUD,COURSE])
f2 = (R|(s,Latin) ∈ R[STUD,COURSE])
h = (R|(s,Math) and (s,Latin) ∈ R[STUD,COURSE])

We have that :

h = f1 ∧ f2 and f1 ∿ f2 mod h

It follows that we can update D1 and D2 independently, as long as we dont change D3 □

Let us see now some examples of decompositions which are not weakly independent.

Example 5

Relation schemas : R(STUD,EXAM,HOUR), R1(STUD,EXAM), R2(EXAM,HOUR)
R3(EXAM)

Integrity constraints : c1 : STUD,HOUR → EXAM in R,
c2 : EXAM → HOUR in R, C = c1 ∧ c2

Data base variables : D = {R}, D1 = {R1}, D2 = {R2}, D3 = {R3}

Mappings : f1 = R[STUD,EXAM], f2 = R[EXAM,HOUR], h = R[EXAM]

We have that h = f1 ∧ f2 but f1 ∿ f2 mod h is not true. That is, the decomposition (f1,f2) is not weakly independent. Therefore, it is not possible to know what updates we can perform on R1 without looking at R2. This, however, does not exclude the existence of such updates. For example, deletion of a tuple in R1, such that the list of exams is not changed, can be done without looking at R2 □

Example 6

Relation schemas : R(NAME,ADDR,TEL), PARISIAN(NAME,ADDR,TEL)
PROVINCIAL(NAME,ADDR,TEL)

Integrity constraints : C : NAME → ADDR,TEL in R

Data base variables : D = {R}, D1 = {PARISIAN}, D2 = {PROVINCIAL}

Mappings : f1 = (R|ADDR = Paris), f2 = (R|ADDR ≠ Paris)

We have that : $f1 \wedge f2 \equiv 0_S$ but f1 and f2 are not independent (they are related by the fact that PARISIAN ∩ PROVINCIAL = ∅). Therefore, given the value of PARISIAN we do not know the set of updates that we can perform without consulting the value of PROVINCIAL □

So far we have studied decomposition in terms of updates of the components. Let us study them now in terms of integrity constraints.

Definition 5 - Let S1 = S<D1|C1> and S2 = S<D2|C2> be two data base state spaces. Let h1 and h2 be two mappings on these spaces, respectively, such that h1(S1) = h2(S2). The join of S1 and S2 with respect to h1 and h2, denoted, S1[h1 = h2]S2, is the following data base state space

$$S<D1 \cup D2 | C1 \wedge C2 \wedge ((h1(D1_t) = h2(D2_t))>$$ □

It should be noted that a "state space join" simply selects all pairs of joinable relations from the given state spaces S1 and S2. Thus, if all possible pairs are joinable (the case where |h1(S1)| = |h2(S2)| = 1), we obtain the cartesian product of S1 and S2.

Example 7

Relation schemas : R1(EMPL,DEPT), R2(DEPT,MGR), R'(DEPT)

Integrity constraints : C1 : EMPL → DEPT in R1, C2 : DEPT → MGR in R2

Data base variables : D1 = {R1}, D2 = {R2}, D' = {R'}

State spaces : S1 = S<D1|C1> , S2 = S<D2|C2> , S' = S<D'|λ>

Mappings : $h1 : S1 \rightarrow S'$ such that $h1(D1_t) = R1[DEPT]$
$h2 : S2 \rightarrow S'$ such that $h2(D2_t) = R2[DEPT]$

As $h1(S1) = h2(S2)$, it follows from Definition 4 that

$$S1[h1=h2]S2 = S < D1 \cup D2 | C1 \wedge C2 \wedge (h1(D1_t) = h2(D2_t)) >$$

$$= S < \{R1,R2\} | C1 \wedge C2 \wedge (R1_t[DEPT] = R2_t[DEPT]) > \quad \square$$

Using the concept of state space join, we can state the following theorem which is an immediate consequence of the definitions.

Theorem 5 - Given a decomposition (f1,f2) let f1', f2' be the mappings such that $f1 \wedge f2 = f1'f1 = f2'f2$.
The following statements are equivalent

(1) (f1,f2) is weakly independent

(2) $S12 = S1[h1=h2]S2$

(3) $C12 = C1 \wedge C2 \wedge (h1(D1_t) = h2(D2_t))$ $\square$

This theorem shows how our definition of weakly independent decompositions (Definition 4) which is valid for

1 - General data base state spaces

2 - General data base mappings

reduces to Rissanen's definition[24] when we consider

1' - State spaces defined by functional dependencies

2' - Projections as data base mappings

We have thus seen the importance of independence in the context of data base decomposition. That is, we cannot update the two components independently unless the mappings producing them are themselves independent. We have also seen that independence is easily decidable in the case of functional dependencies and projections. It would be interesting to study more general cases.

VIEW UPDATING

The view mechanism is a facility offered by most data base systems. It was introduced for two reasons.

1 - Some times the user does not need the whole amount of information stored in the data base. That is, he may not need some of the attributes in a relation or some parts of a relation or even some whole relations altogether. In such cases, having to cope with the whole of the data base is annoying and confusing.

2 - In the context of specific applications, the user may need a structure on the information different than the one provided in the data base.

To satisfy these requirements we must define (a) a new data base variable describing the user needs and (b) a mapping which, at each time t, can compute the value of this new variable in terms of the data base value at time t. This leads to the following formal definition of a view.

Definition 5 - Let $S = S<D|C>$ be a given data base state space. A view is a pair (D',f) where D' is a data base variable and f is a mapping from S into $S<D'|\lambda>$. □
The mapping f defines a data base schema $<D'|C'>$ that we shall call the user subschema.

Example 8

Relation schemas : R(EMPL,SAL,ADDR,DEPT,MGR,$\#$EMPL),
R'(EMPL,SAL,ADDR)

Integrity constraints : c1 : EMPL $\rightarrow$ SAL,ADDR,DEPT in R
c2 : DPT $\rightarrow$ MGR,$\#$EMPL in R, C = c1 $\wedge$ c2, C' : EMPL $\rightarrow$ SAL,ADDR in R'

Data base variables : D = {R}, D' = {R'}

Mappings : f = (R|DEPT = Sales)[EMPL,SAL,ADDR]

The pair (D',f) is a view and the data base schema $< D' | C' >$ is the user subschema. In the user subschema we replace, in a sense, the mapping f, that the user does not have to know, by a constraint C', that the user can see. In this way, we offer the user a new data base tailored to his own needs. □

Two methods are possible in order to implement views.

1 - we store-physically-the value $f(D_t)$

2 - we store only the definition of f and compute the value $f(D_t)$ whenever this is necessary.

The first method allows fast execution of user queries directly on $f(D_t)$, but it presents multiple disadvantages

1 - Waste of memory space

2 - Redundancy : the common part of D_t and $f(D_t)$ is stored twice

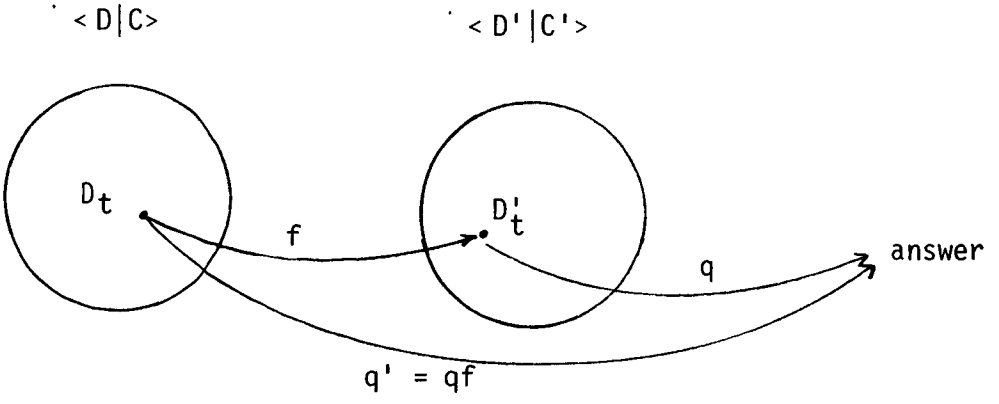
3 - Consistency : every time the data base is updated we must propagate the changes to the view

Because of these disadvantages, implementation of views is generally done by storing only the definition of f.

View users interact with their subschemas by issuing queries and update requests. Let us see how a data base system might handle this interaction and what kind of problems may appear.

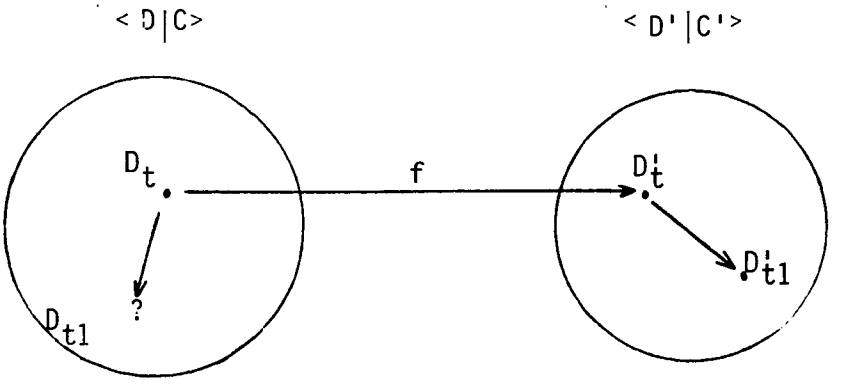
VIEW QUERIES : A query on the view defines a mapping on the set $S < D' | C' >$. We have

$$q(D'_t) = q\,(f(D_t)) = qf(D_t)$$



Therefore, any query q on the view can be "translated" into a query q' = qf on the data base in a straightforward manner.

VIEW UPDATES : The situation is now more delicate. The user wishes to change the current value D'_t into a new one, say D'_{t1}. As D'_t does not exist physically, we must change the current data base value D_t to some new value D_{t1}. What this new value should be ?



The answer to this question involves such practical and theoretical problems that no existing data base system allows view updating (except in the very special case where the view is identical to one of the data base relations). We shall illustrate the problems involved using examples.

Example 9 : UNDEFINED VALUES

Relational schemas : R(NAME,ADDR,TEL), R'(NAME,ADDR)

Integrity constraints : C : NAME → ADDR,TEL in R

Data base variables : D = {R}, D' = {R'}

Mappings : f = R[NAME,ADDR]

Views : (D',f)

View updates : u : Insert the tuple (John,Paris)

The new data base value D_{t1} will have a tuple (John,Paris,-) with an undefined value for TEL □

Example 10 : MULTIPLE CHOICE

Relational schemas : R(MGR,DEPT,SECR), R'(MGR,DEPT)

Integrity constraints : C : MGR ↔ DEPT ↔ SECR in R

Data base variables : D : {R}, D' = {R'}

Mappings : f = R[MGR,DEPT]

Views : (D',f)

View updates : u : Change the tuples (John,Sales) and (Peter,Toys) to (John,toys) and (Peter,Sales).

In looking for the new data base value D_{t1} we have a choice in this case :

1 - we exchange the managers of "toys" and "sales" and the secretaries stay with the departments

2 - we exchange the managers of "toys" and "sales" and the secretaries follow their managers.

Both are legal updates of the data base and both reflect faithfully the view update u □

Example 11 : SIDE EFFECTS

Relational schemas : R1(EMPL,SAL,DEPT), R2(DEPT,MGR, # EMPL)
R'(EMPL,SAL,DEPT)

Integrity constraints : c1 : EMPL → SAL,DEPT in R1, c2 : DEPT → MGR in R2, c3 : # EMPL in R2 is the number of employees in the corresponding department, computed from R1, C : c1 ∧ c2 ∧ c3

Data base variables : D = {R1,R2}, D' = {R'}

Mappings : f = R1

Views : (D',f)

View updates : u : Insert the tuple (John,10000,Sales)

This view update can be reflected easily at the data base level by simply inserting the tuple (John,10000,Sales) into R1. However, the number of employees in "Sales" has now increased by 1. So unless we reflect this change in R2, we violate c3 ! □

Example 12 : CONSTRAINT VIOLATION

Relational schemas : R(STUD,EXAM,HOUR), R'(STUD,EXAM)

Integrity constraints : c1 : STUD,EXAM → HOUR in R, c2 : EXAM → HOUR in R, C = c1 ∧ c2

Data base variables : D = {R}, D' = {R'}

Mappings : f = R[STUD,EXAM]

Views : (D',f)

View update : u : Insert the tuple (John,Math)

If the Math-exam is already present in the data base then there is some hour H already assigned to it. So all we have to do is to insert the tuple (John,Math,H) in R. But, if John has already registered for another exam taking place at H, then we violate c1 (in this case, it is impossible to reflect the view update at the data base level) □

Before we discuss a solution to the view updating problem let us give a formal definition of an update.

Definition 6 - Given a data base schema $<D|C>$, an update is a mapping $u : S<D|C> \rightarrow S<D|C>$ □

It should be noted that, according to this definition, the result of an update always satisfies the integrity constraints. Let us see an example.

Example 13

Relational schemas : R(NAME,AGE)

Integrity constraints : C : NAME → AGE

Data base variables : D = {R}

"Insert (John,25)" is not on update because to any value D_t containing the tuple (John,20) it associates a value D_{t1} that violates C. On the other hand "Insert (John,25), if John does not appear in the data base", is an update □

The set of updates that a user is allowed must satisfy some minimal requirements that we define now.

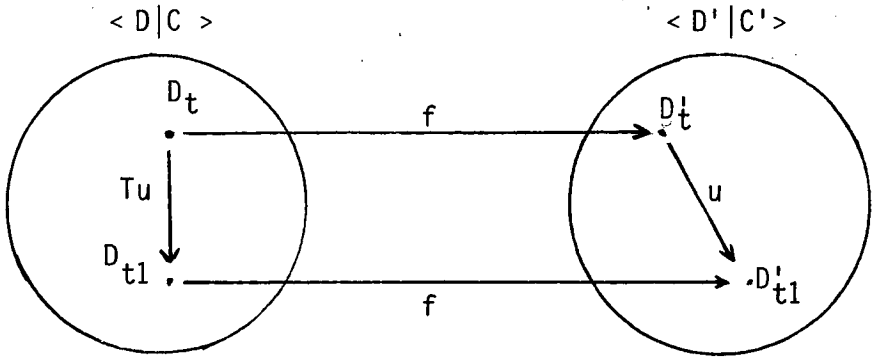
Definition 7[4] - Let $S = S<D|C>$. A set UPD of updates on S is called complete iff

(i) UPD is closed under (finite) composition

(ii) $\forall\ D_t \in S \quad \forall\ u \in UPD\ \exists\ u' \in UPD$ such that $u'u(D_t) = D_t$ □

The fact that UPD is closed under composition means simply that if the user is allowed two updates u_1 and u_2 then he can apply them in any order he wants. The second property of the set UPD is an "error erasing" capability. That is, the user can always come back to the state of the data base before the update. This does not mean that u' is the inverse of u. For example, a set of updates consisting of insertion and deletion of tuples in a relation is a complete set (although deletion is not inversible).

Let us go back again to the problem of translating view updates. The view being at a state $D'_t = f(D_t)$, it is taken by u to a new state $D'_{t1} = u(D'_t) = uf(D_t)$. What we are looking for is a data base update



T_u that takes the data base from D_t to D_{t1}. The problem is to find D_{t1}. One obvious requirement for D_{t1} is that it must map under f onto the updated view. That is,

$$f(D_{t1}) = D'_{t1} \quad \text{or}$$
$$f(T_u(D_t)) = u(f(D_t))$$

As this last equality must hold for every D_t in S, we obtain

$$fT_u = uf$$

Unfortunately, this condition alone is not sufficient to define the translating update T_u. This is so because f extracts just a part of the information in the data base. Therefore, there is, in general, more than one data base value that correspond to the same view value. Of course, in most concrete examples one can use common sense to distinguish a "good" from a "bad" translation. Let us see an example.

Example 14

Relation schemas : R(EMPL,DEPT,MGR), R'(EMPL,DEPT)

Integrity constraints : c1 : EMPL → DEPT in R, c2 : DEPT → MGR in R,
C = c1 ∧ c2

Data base variables : D = {R}, D' = {R'}

Mappings : f = R[EMPL,DEPT]

Views : (D',f)

View updates : u : If Sales ∈ R'[DEPT] insert (Mike,Sales) in R'.
Suppose the data base value and the corresponding view value are as follows

D_t

EMPL	DEPT	MGR
John	Sales	Nick
Peter	Sales	Nick
Jack	Toys	Tim

D'_t

EMPL	DEPT
John	Sales
Peter	Sales
Jack	Toys

We show below two data base values D_{t1} and D_{t2} that produce the same updated view $D'_{t1} = u(D'_t)$.

D_{t1}

EMPL	DEPT	MGR
John	Sales	Nick
Peter	Sales	Nick
Jack	Toys	Tim
Mike	Sales	Nick

D_{t2}

EMPL	DEPT	MGR
John	Sales	Tim
Peter	Sales	Tim
Jack	Toys	Nick
Mike	Sales	Tim

D'_{t1}

EMPL	DEPT
John	Sales
Peter	Sales
Jack	Toys
Mike	Sales

"Good sense" says that between D_{t1} and D_{t2} only the first is good. □

Is there any formal way to characterize good translations of view updates ? We suggest the following minimal requirement for a translation to be good.

"If the update changes nothing in the view then nothing must change in the data base".

Thus, in the previous example, if sales $\notin$ R'[DEPT] then nothing changes in the view. It seems then reasonable that nothing must change in the data base either. This is why D_{t1} was chosen as the "good" one. Hence the following definition of a (good) translation

Definition 8[4] - Let (D',f) be a view of $S = S<D|C>$. Let u be a view update. The data base update T_u is a translation of u iff

(i) $\quad uf = fT_u$

(ii) $\quad uf(D_t) = f(D_t) \Rightarrow T_u(D_t) = D_t \qquad \forall\ D_t \in S$ □

This definition states the minimal requirements for the translation of a view update. We shall show next that these requirements are also sufficient. That is, we shall show that every translation of a view update that satisfies Definition 8 makes sense. First, however, recall that we are dealing not with just one view update but with a set of view updates (that we have assumed complete). Therefore, we need a "translator" which associates a translation to every view update in the set. We shall use the symbol U_f to denote the set of all updates on a view defined by the mapping f on $S<D|C>$. That is,

$$U_f = \{u | u : f(S) \to f(S)\}$$

Following this notation, U_1 denotes the set of all updates on $S<D|C>$.

Definition 9[4] - Given $S<D|C>$ and a view (D',f), let $U \subset U_f$ be a complete set of view updates. A translator for U is a mapping $T : U \to U_1$ such that

(i) $\quad \forall\ u \in U,\ T(u)$ is a translation of u

(ii) $\quad \forall\ u_1, u_2 \in U,\ T(u_1u_2) = T(u_1)T(u_2)$ □

Essentially, this definition says that the result of a sequence of view updates is the same whether they are executed "en block" or separately. There are two important consequences of this definition

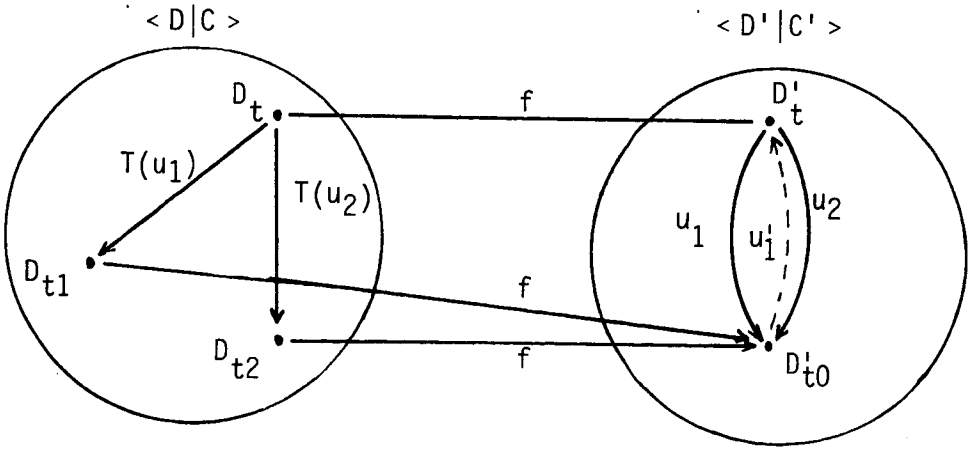
1 - If $1_{S'} \in U_f$ then $T(1_{S'}) = 1_S$, where $S' = S\langle D'|C'\rangle$. That is, if nothing changes in the view then nothing changes in the data base either (precisely as it is required by Definition 8)

2 - The translation of a view update depends only on the initial and the final state and not on the nature of the update that caused the change (as the following theorem shows)

Theorem 6 - Given are : $\langle D|C\rangle$, a view (D',f), a complete set $U \subset U_f$, a translator T for U and two view updates $u_1, u_2 \in U$. The following is true

$$u_1 f(D_t) = u_2 f(D_t) \Rightarrow T(u_1)(D_t) = T(u_2)(D_t) \qquad \square$$

Proof : Let $D'_t = f(D_t)$, $D_{t1} = T(u_1)(D_t)$, $D_{t2} = T(u_2)(D_t)$, $D'_{t0} = u_1(D'_t) = u_2(D'_t) = f(D_{t1}) = f(D_{t2})$



As the set U is complete, there exists $u'_1 \in U$ such that $u'_1(D'_{t0}) = D'_t$. Therefore, we have

$$u'_1 u_1 f(D_t) = f(D_t) \Rightarrow T(u'_1 u_1)(D_t) = D_t \Rightarrow T(u'_1)[T(u_1)(D_t)] = D_t$$
$$\Rightarrow T(u'_1)(D_{t1}) = D_t$$
$$u'_1 u_2 f(D_t) = f(D_t) \Rightarrow T(u'_1 u_2)(D_t) = D_t \Rightarrow T(u'_1)[T(u_2)(D_t)] = D_t$$
$$\Rightarrow T(u'_1)(D_{t2}) = D_t$$

It follows that

$$T(u_1')(D_{t1}) = T(u_1')(D_{t2}) = D_t \qquad (1)$$

Also, we have

$$u_1u_1'f(D_{t1}) = f(D_{t1}) \Rightarrow T(u_1u_1')(D_{t1}) = D_{t1} \Rightarrow T(u_1)[T(u_1')(D_{t1})] = D_{t1}$$
$$\Rightarrow T(u_1)(D_t) = D_{t1} \quad \text{(because of (1))} \qquad (2)$$
$$u_1u_1'f(D_{t2}) = f(D_{t2}) \Rightarrow T(u_1u_1')(D_{t2}) = D_{t2} \Rightarrow T(u_1)[T(u_1')(D_{t2})] = D_{t2}$$
$$\Rightarrow T(u_1)(D_t) = D_{t2} \quad \text{(because of (1))} \qquad (3)$$

It follows from (2) and (3) that $D_{t1} = D_{t2}$, Q.E.D.

Now that we have defined formally the notion of a translator the question is how to construct one. A method for constructing translators is presented elsewhere[4]. What we would like to do here is to describe briefly the approach and show how our results on independence (see Part 1 of this paper) can be applied in this context.

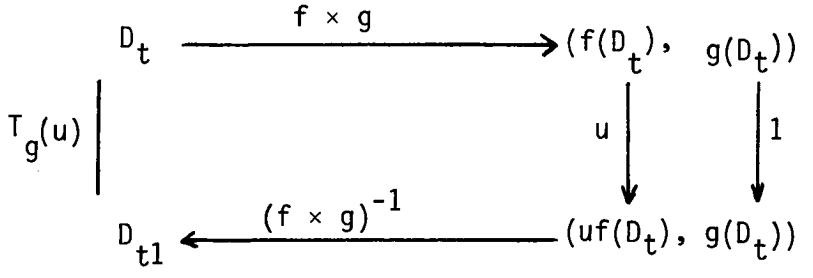
Given are

1 - A data base schema $<D|C>$

2 - Two views (D',f) and (D'',g) such that the mappings f and g are complementary

3 - An update $u \in U_f$

Suppose the data base is at some state D_t. An equivalent representation of D_t is the pair $(f(D_t),g(D_t))$. If we apply the update u on $f(D_t)$ we move to the new pair $(uf(D_t),g(D_t))$. As f and g are complementary, we can apply the function $(f \times g)^{-1}$ to the new pair in order to find the corresponding data base state.



This process defines the following translation for u.

$T_g(u) = (f \times g)^{-1}(uf \times g)$

However, an important question remains. Before we apply the function $(f \times g)^{-1}$ we must make sure that the new pair $(uf(D_t), g(D_t))$ corresponds to a data base state. Hence the following definition.

Definition 10 - Let $<D|C>$ be a data base schema. Let (D', f) and (D'', g) be two views such that f and g are complementary. Let $u \in U_f$

(i) the update u is called translatable under constant g (g-translatable, for short) if $(uf \times g)(S) \subset (f \times g)(S)$

(ii) If u is g-translatable, its translation under constant g (g-translation, for short), denoted by $T_g(u)$, is defined as follows $T_g(u) = (f \times g)^{-1}(uf \times g)$ □

It is interesting to note that

$uf \times g = (u \times 1)(f \times g)$

and, therefore, the g-translation of u can be written as follows $T_g(u) = (f \times g)^{-1}(u \times 1)(f \times g)$

This reminds one of a base change of a linear mapping :

- $f \times g$ stands for the base-change matrix (which takes us from a representation D_t to a new one $(f(D_t), g(D_t))$).
- $(f \times g)^{-1}$ stands for the inverse of the above matrix
- $u \times 1$ stands for a diagonal matrix

Going back to definition 10 we would like to comment briefly on some important results discussed elsewhere[4,27]

1 - If u is g-translatable then the mapping $T_g(u)$ satisfies definition 8 and, therefore, it is a translation. Moreover, if g' is a complement of f and $g' \leq g$ then u is g'-translatable and $T_{g'}(u) = T_g(u)$

2 - Let $U \subset U_f$ be a complete set. Let g be a complement of f such that every $u \in U$ is g-translatable. Then the mapping $T_g : U \to U_1$ such that $T_g(u) = (f \times g)^{-1}(uf \times g)$ is a translator for U.

3 - The method described in 2 for constructing translators of complete sets is unique. That is, given any translator T for a complete set U, we can find a complement g of f such that

a Every $u \in U$ is g-translatable

b $T_g = T$

These basic results on translation of view updates rely on g-translatability for which no characterization is given[4]. We conclude this section by an application of our results on independence to g-translatability.

Theorem 7 - Given a data base schema <D|C> and two complementary mappings f and g, the following are equivalent

(1) $\forall u \in U_f$, u is g-translatable

(2) $\forall v \in U_g$, v is f-translatable

(3) $f \sim g$ □

Proof : (3) ⟹ (1)

$$(uf \times g)(S) = ((u \times 1)(f \times g))(S)$$
$$= (u \times 1)((f \times g)(S))$$
$$= (u \times 1)(f(S) \times g(S)) \quad \text{[because } f \sim g\text{]}$$
$$= (uf(S)) \times g(S) \subset f(S) \times g(S) = (f \times g)(S) \quad \text{Q.E.D.}$$

(1) ⟹ (3)

Let $A(u) = (uf \times g)(S)$. It follows from (1) that

$A(u) \subset (f \times g)(S) \quad \forall u \in U_f$ (a)

Let $A = \cup \{A(u) | u \in U_f\}$. It follows from (a) that

$A \subset (f \times g)(S)$ (b)

On the other hand, we have :

$$A = \{(uf(D_t),\ g(D_t)) \mid D_t \in S,\ u \in U_f\}$$
$$= \{(f(D_{t'}),\ g(D_t)) \mid D_t \in S,\ D_{t'} \in S\}$$
$$= f(S) \times g(S) \qquad (c)$$

It follows from (b) and (c) that $f(S) \times g(S) \subset (f \times g)(S)$. Therefore, $(f \times g)(S) = g(S) \times g(S)$, Q.E.D.

The condition $f \sim g$ in the above theorem is rather strong, as it implies that $f \wedge g \equiv 0_S$. If this is not the case, we would still like to know what is the maximal set of g-translatable updates.

Theorem 8 - Given a data base schema $<D|C>$ and two complementary mappings f and g, let h be a mapping such that $f \wedge g = hf$. Then, $\forall\ u \in U_f$, u g-translatable $\Longrightarrow$ $hu = h$ □

Proof : As u is g-translatable we have that : $\forall\ D_t \in S \quad \exists\ D_{t'} \in S$ such that

$$f(D_{t'}) = uf(D_t) \qquad (1)$$

$$g(D_{t'}) = g(D_t) \qquad (2)$$

It follows from (2) that : $(f \wedge g)(D_{t'}) = (f \wedge g)(D_t)$. Therefore,

$$hf(D_{t'}) = hf(D_t) \qquad (3)$$

It follows from (1) and (3) that

$$h(uf(D_t)) = h(f(D_{t'})) = hf(D_t)$$

Therefore, $hu = h$ Q.E.D.

Thus, we have determined an upper bound for the set of g-translatable updates. It consists of all updates that leave the "common part" of f and g invariant. Under what conditions this upper bound is attained ? The following theorem provides the answer.

Theorem 9 - Given a data base schema $< D|C >$ and two complementary mappings f and g, let h be a mapping such that $f \wedge g = hf$. The following statements are equivalent.

(1) $hu = h \Longrightarrow u$ is g-translatable, $u \in U_f$

(2) $f \sim g \bmod f \wedge g$ □

Proof : (1) ⟹ (2)

Let $D_t \in S$. Define the set

$B = \{D_{t'} | f \wedge g(D_{t'}) = f \wedge g(D_t)\}$

It is enough to show that : $(f \times g)(B) = f(B) \times g(B)$. It follows from (1) that $\forall\, u \in U_f$ such that $huf(D_t) = hf(D_t)$, $(uf(D_t), g(D_t)) \in (f \times g)(S)$

In other words

$\forall\ D_{t'}, D_t$ such that $hf(D_{t'}) = hf(D_t)$, $(f(D_{t'}), g(D_t)) \in f \times g)(S)$

As $hf = f \wedge g$, it follows that

$\forall\ D_{t'}, D_t$ such that $f \wedge g(D_{t'}) = f \wedge g(D_t)$, $(f(D_{t'}), g(D_t)) \in (f \times g)(S)$

Then, the definition of B above implies that

$f(B) \times g(B) \subset (f \times g)(B)$

Therefore, $(f \times g)(B) = f(B) \times g(B)$, Q.E.D.

(2) ⟹ (1) The proof of this part can be done by simply inversing the steps of the first part of the proof □

DATA BASE MAPPINGS AND THE THEORY OF RELATIONS

Whereas the theory of relations studies the structure of individual relations, data base mappings deal with sets of relations (i.e., state spaces) and as such they are on a higher level of abstraction. Nevertheless, there are some parallels between the two theories that we summarize in the following table.

RELATION	STATE SPACE
Functional Dependency $X \rightarrow Y$	Dominance $f \geq g$
Multivalued Dependency $X \twoheadrightarrow Y \mid Z$	weak Independence $f \sim g \bmod f \wedge g$
Equijoin $R(XY)[h_1 = h_2]R(ZW)$	Equijoin $S' < D1 \mid C1 > [h_1 = h_2] S < D2 \mid C2 >$
Cross dependency $x(X,Y)$	Total independence $f \sim g$
Cartesian product $R(X) \times R(Y)$	Cartesian product $S < C1 \mid D1 > \times S < D2 \mid C2 >$

Let us look closer into each case :

1) Funtional Dependency/Dominance

$X \rightarrow Y$ iff $\forall x_1, x_2 \in R$, $x_1.X = x_2.X \Rightarrow x_1.Y = x_2.Y$

$f \geq g$ iff $\forall D_{t1}, D_{t2} \in S$; $f(D_{t1}) = f(D_{t2}) \Rightarrow g(D_{t1}) = g(D_{t2})$

The two definitions are isomorphic, the only difference being that a relation must be finite whereas a state space need not be finite. It is interesting to note that the following axiom system is complete for functional dependencies.

FD1 : $X \rightarrow Y$

FD2 : $X \rightarrow Y$ and $Y \rightarrow Z \Rightarrow X \rightarrow Z$

FD3 : $X \rightarrow Y$ and $X' \rightarrow Y' \Rightarrow X \cup X' \rightarrow Y \cup Y'$

FD4 : $X \rightarrow Y$ and $X \subset X'$ and $Y' \subset Y \Rightarrow X' \rightarrow Y'$

On the other hand, the following "similar" axiom system is satisfied by the dominance relation (proving its completeness is a different story).

D1 : $f \geq f$

D2 : $f \geq g$ and $g \geq h \Rightarrow f \geq h$

D3 : $f \geq g$ and $f' \geq g' \Rightarrow f \times f' \geq g \times g'$

D4 : $f \geq g$ and $f_1 = f \times f_2$ and $g = g_1 \times g_2 \Rightarrow f_1 \geq g_1$

2) Multivalued Dependency/Weak Independence

The multivalued Dependency $X \twoheadrightarrow Y$ is true in R(XYZ) iff

$$\left.\begin{array}{l} u_1 \in R(XYZ) \\ u_2 \in R(XYZ) \\ u_1.X = u_2.X \end{array}\right\} \Rightarrow \exists\, u_o \in R(XYZ) \text{ such that } \left\{\begin{array}{l} u_o.X = u_1.X = u_2.X \\ u_o.Y = u_1.Y \\ u_o.Z = u_2.Z \end{array}\right.$$

The weak independence $f \sim g \bmod f \wedge g$ is defined as follows ($f \wedge g = h$) :

$$\left.\begin{array}{l} D_{t1} \in S \\ D_{t2} \in S \\ h(D_{t1}) = h(D_{t2}) \end{array}\right\} \Rightarrow \exists\, D_{to} \in S \text{ such that } \left\{\begin{array}{l} h(D_{to}) = h(D_{t1}) = h(D_{t2}) \\ f(D_{to}) = f(D_{t1}) \\ g(D_{to}) = g(D_{t2}) \end{array}\right.$$

3) Cross Dependency/Total Independence

The cross dependency $x(X,Y)$ holds in R(XY) iff

$$\left.\begin{array}{l} u_1 \in R(XY) \\ u_2 \in R(XY) \end{array}\right\} \Rightarrow \exists\, u_o \in R(XY) \text{ such that } \left\{\begin{array}{l} u_o.X = u_1.X \\ u_o.Y = u_2.Y \end{array}\right.$$

The total independence between f and g is defined as follows :

$$\left.\begin{array}{l} D_{t1} \in S \\ D_{t2} \in S \end{array}\right\} \Rightarrow \quad D_{to} \in S \text{ such that } \left\{\begin{array}{l} f(D_{to}) = f(D_{t1}) \\ g(D_{to}) = g(D_{t2}) \end{array}\right.$$

4) Equijoin of relations / Equijoin of spaces

The equijoin between two relations is defined as follows

$$R_1(XY)[Y=Z]R_2(ZW) = \{uv \mid u \in R_1,\ v \in R_2,\ u.Y = v.Z\}$$

The equijoin between two state spaces is defined as follows

$$S_1[h_1=h_2]S_2 = \{(D1_t, D2_t) \mid D1_t \in S1,\ D2_t \in S2,\ h_1(D1_t) = h_2(D2_t)\}$$

Moreover, there is a similarity between the theorems that relate equijoin of relations and multivalued dependency, on the one hand, and equijoin of schemas and independence on the other hand.

Theorem : $R(XY)[Y=Y]R(YZ) = R(XYZ)$ iff $X \twoheadrightarrow Y$ □

Theorem : $S = f(S)[h_1=h_2]g(S)$ iff $f \sim g \bmod f \wedge g$ $(fh_1 = gh_2 = f \wedge g)$ □

Similarly, in the degenerate cases we have :

$$R(XY) = R(X) \times R(Y) \text{ iff } x(X,Y)$$

$$S = f(S) \times g(S) \text{ iff } f \sim g$$

To finish with parallels, we shall conclude this section by showing a last one between our definition of independence and that of random variables in probability theory.
Let us first recall briefly some basic definitions.
Let S be a sample space and ε an event space on S. A probability measure is any function $p : \varepsilon \to [0,1]$ such that

1 - $p(E_1 \cup E_2) = p(E_1) + p(E_2)$ if $E_1 \cap E_2 = \emptyset$

2 - $p(S) = 1$

3 - $p(\emptyset) = 0$

The triple (S,ε,p) is called a probability space. Given two probability spaces (S,ε,p) and (S',ε',p'), a random variable on S is any function $f : S \to S'$ such that $f^{-1}(E') \in \varepsilon$, for every $E' \in \varepsilon$. The distribution of f, denoted p_f, is a probability measure on S defined as follows $p(E) = p(f^{-1}(E'))$, $\forall\ E' \in \varepsilon'$

Two random variables f and g are called independent iff

$$p(f^{-1}(A) \cap g^{-1}(B)) = p(f^{-1}(A))p(g^{-1}(B)) \quad \forall\ A \in \varepsilon_f \quad \forall\ B \in \varepsilon_g$$

Suppose now that our sample space S is the state space of a data base $<D|C>$, that is, suppose $S = S^{\cdot}<D|C>$. To simplify matters, suppose S is finite and that ε is the set of all subsets of S. Further, suppose p is such that $p(E) = |E|/|S|$, for every $E \subset S$. Consider next two data base mapping f and g on S. These mappings can be viewed either as data base mappings or as random variables on S. The following theorem shows that the definitions of independence in the two cases are equivalent.

Theorem 10 - Let S, ε, p, f, g be as defined above. If f and g are complementary data base mappings then the following statements are equivalent.

(1) The data base mappings f and g are independent
(2) The random variables f and g are independent □

Proof : We show first that

$$\forall\ A \subset f(S) \quad \forall\ B \subset g(S) \qquad |f^{-1}(A) \cap g^{-1}(B)| = |A||B| \tag{a}$$

$$\begin{aligned} f^{-1}(A) \cap g^{-1}(A) &= \{D_t | f(D_t) \in A,\ g(D_t) \in B\} \\ &= \{D_t | (f(D_t),\ g(D_t)) \in A \times B\} \\ &= \{D_t | (f \times g)(D_t) \in A \times B\} \\ &= (f \times g)^{-1}(A \times B) \end{aligned}$$

As $f \times g$ is injective, it follows that

$$|(f \times g)^{-1}(A \times B)| = |A \times B| = |A||B|$$

and this implies (a)

$$(1) \Rightarrow (2)$$

We must show that

$$\forall A \subset f(S) \quad \forall B \subset g(S) \quad p(f^{-1}(A) \cap g^{-1}(B)) = p(f^{-1}(A))\, p(g^{-1}(B)) \qquad (b$$

It follows from (a) that

$$p(f^{-1}(A) \cap g^{-1}(B)) = \frac{|A||B|}{|S|} \qquad (c$$

On the other hand, we have

$$p(f^{-1}(A)) = |f^{-1}(A)|/|S| = |f^{-1}(A) \cap S|/|S| = |f^{-1}(A) \cap g^{-1}(g(S))|/|S|$$

$$= |A||g(S)|/|S| \qquad \text{(using (a))} \qquad (d$$

Similarly, we obtain

$$p(g^{-1}(A)) = |f(S)||B|/|S| \qquad (e$$

As $f \sim g$ and $f \times g$ is injective, we obtain

$$|f \times g(S)| = |f(S) \times g(S)| = |S|$$

Therefore,

$$|S| = |f(S)|\,|g(S)| \qquad (k$$

Using (c), (d), (e) and (k) we can check that (b) holds Q.E.D.

$$(2) \Rightarrow (1)$$

As f and g are independent random variables (b) holds. It follows, that

$$\forall D'_t \in f(S) \quad \forall D''_t \in g(S) \quad f^{-1}(D'_t) \cap g^{-1}(D''_t) \neq \emptyset$$

and this implies that the data base mappings f and g are independent (theorem 9 of Part 1 of this paper), Q.E.D.

Finally, let us note that $f \sim g \bmod h$ can be interpreted as contitional independence of random variables in a similar manner.

CONCLUSION

In this second part of the paper, we have seen two important applications of the theory of data base mappings. First, to data base decompositions, where general definitions where given for

- lossless decompositions
- independent decompositions
- weakly independent decompositions

We have shown that our definitions reduce to well known definitions in the special case of projections and functional dependencies.

Second, we have applied our theory to view updating. The notion of weak independence was used to characterize the set of view updates that are translatable under a given complement.

We believe that these applications provide a (a posteriori) justification for a theory of data base mappings.

Finally, we have discussed some parallels between the theory of data base mappings, on the one hand, and the theory of relations and probability theory, on the other hand.

Most of the results that we have seen in both parts of this paper are in some sense extensions of similar results in the theory of relations. Of course, this comes as no surprise. We have defined a data base state space to be a set and a data base mapping to be a function on this set. Therefore, there is nothing "relational" in it. In fact, this is simply a reminder that the theory of relations is just a special case of the theory of functions.

REFERENCES

[1] Aho A.V., C. Beeri et J.D. Ullman (1978). "The theory of joins in relational database", submitted to TODS. A preliminary version appeared in Proc. Eighteenth Annual IEEE Symposium on Foundations of Computer Science, pp. 107-113.

[2] Armstrong W.W. (1974). "Dependency structures of data base relationships", Proc. 1974 IFIP Congress, pp. 580-583, North Holland, Amsterdam.

[3] Bancilhon F. (1978). "On the completeness of query languages for relational databases", Proc. Seventh Symp. on Mathematical Foundations of Computer Science, Springer Verlag.

[4] Bancilhon F., N. Spyratos. "Update Semantics of Relational Views", To appear in ACM TODS.

[5] Beeri C., P.A. Bernstein et N. Goodman (1978). "A sophisticate's introduction to database normalization theory", Proc. ACM Intl. Conf. on Very Large Data Bases, pp. 113-124.

[6] Beeri C., A.O. Mendelzon, Y. Sagiv et J.D. Ullman (1979). "Equivalence of relational database schemes", Proc. Eleventh Annual ACM Symposium on the Theory of Computing, pp. 319-329.

[7] Beeri C., J. Rissanen (1979). "On the decomposition of relational Data Base Schemas", Acte des Journées d'étude "Bases Formelles pour Bases de Données", Toulouse.

[8] Biskup J., U. Dayal et P.A. Bernstein (1979). "Synthesizing independent database schemas", ACM/SIGMOD International Symposium on Management of Data, pp. 143-152.

[9] Chamberlain D.D., J.N. Gray et D.D. Traiger (1975). "Views, authorization and locking in a relational data base system". Proceedings of AFIPS NCC, Vol. 44.

[10] Chandra A.K. et D. Harel (1978). "Computable queries for relational databases", Proc. Eleventh Annual ACM Symposium on the Theory of Computing, pp. 309-319.

[11] Codd E.F. (1970). "A relational model for large shared data banks", Comm. ACM 13 : 6, pp. 377-387.

[12] Codd E.F. (1972a). "Further normalization of the data base relational model", in Data Base Systems (R. Rustin, ed.) Prentice Hall, Englewood Cliffs, N.J., pp. 33-64.

[13] Codd E.F. (1972b). "Relational completeness of data base sub-languages", ibid, pp. 65-98.

[14] Dayal U. et P.A. Bernstein (1978). "On the updatability of Relational views", Proceedings of the 4th International Conference on Very Large Data Bases, West Berlin.

[15] Dayal U. (1979). "Schema-Mapping problems in data base systems", Ph. Thesis, Harvard University.

[16] Delobel C. et R.C. Casey (1972). "Decomposition of a database and the theory of Boolean switching functions", IBM J. Res. 17 : 5, pp. 370-386.

[17] Delobel C. (1978). "Normalization and hierarchical dependencies in the relation data model", ACM Trans. on Database Systems 3 : 3, pp. 201-222.

[18] Fagin R. (1977). "Multivalued dependencies and a new normal form for relational data bases", ACM Trans. on Database Systems 2 : 3, pp. 262-278.

[19] Makinouchi A. (1977). "A consideration on Normal Form of Not-necessarily Normalized Relation in the Relation Data Model". Proceedings Very Large Data Bases, Third International Conf., Tokyo, Japan, October 6-8, pp. 447-453.

[20] Nicolas J.M. (1978). "Mutual dependencies and some results on undecomposable relations", Proc. ACM Intl. Conf. on Very Large Data Bases, pp. 360-367.

[21] Paolini P. et G. Pelegatti (1977). "Formal definition of mappings in a data base", ACM/SIGMOD, Proceedings of the Int. Conf. on Management of Data, August, pp. 40-46.

[22] Paolini P. (1979). "Formal definition of views and verification of applications programs". Acte des journées d'étude "Bases Formelles pour Bases de Données", Toulouse.

[23] Paredaens J. (1978). "On the expressive power of relational algebra", Information Processing Letters 7 : 2, pp. 107-111.

[24] Rissanen J. (1977). "Independent components of relations", ACM Trans. on Database Systems 2 : 4, pp. 317-325.

[25] Rissanen J. (1978). "Relations with functional and join dependencies and their representation by independent components", Unpublished manuscript, IBM, San Jose, California.

[26] Sevcik K.C., A.L. Furtado (1978). "Complete and Compatible sets of Update Operations", Proc. ICMOD 78, Milan, Italie, Juin, pp. 247-270.

[27] Spyratos N. (1980). "Translation Structures of Relational Views", Proc. 6th Intl. Conf. on Very Large Data Bases, Montreal, Canada.