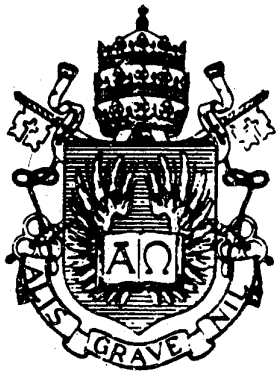


PUC

Series: Monografias em Ciência da Computação
Nº 02/78

PERMITTING UPDATES THROUGH VIEWS OF DATA BASES

by

A. L. Furtado
K. C. Sevcik

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vivente, 225 – ZC-19
Rio de Janeiro – Brasil

Series: Monografias em Ciência da Computação

Nº 02/78

Series Editor: Michael F. Challis January, 1978

PERMITTING UPDATES THROUGH VIEWS OF DATA BASES*

by

A. L. Furtado 1934

K. C. Sevcik**

* This research has been partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) National Research Council of Canada, Canadian International Development Agency, and Financiadora de Estudos e Projetos (FINEP)

** Computer Systems Research Group, Univeristy of Toronto, Canada.

For copies contact

Rosane Teles Lins Castilho
Head, Set. Doc. Inf.
Depto. Informatica - PUC/RJ
Rua Marques de São Vicente, 209 - Gávea
20.000 - Rio de Janeiro - RJ - Brasil

ABSTRACT

Providing different views (logical images of the structure of a data base) to various users creates the problem of determining how update operations expressed in terms of the views should affect the stored form of the data base. For data bases with a relational organization, we indicate the effects of a wide range of update operations on views. We conclude that some operations must be prohibited in order to assure harmonious interactions among data base users, but that many other operations can be allowed even though the structure of the view may differ substantially from the actual structure of the bata base. We consider views not only as "windows" through which to see a data base in a particular way, but also as "shades" to conceal and protect information, and as "screens" to intercept any update operations that could leave the stored form of the data base in an unacceptable state.

KEY WORDS:

Data bases, relational model, update operations, constraints, views, algebra of quotient relations.

RESUMO

Prover diferentes visões (imagens lógicas da estrutura de um banco de dados) para vários usuários cria o problema de determinar como operações de atualização expressas em termos de visões devem afetar a forma armazenada do banco de dados. Para bancos de dados com uma organização relacional, indicamos os efeitos de uma ampla gama de operações de atualização sobre visões. Concluimos que algumas operações devem ser proibidas a fim de assegurar interações harmoniosas entre os usuários do banco de dados, mas que muitas outras operações podem ser permitidas embora a estrutura da visão difira substancialmente da estrutura real do banco de dados. Consideramos visões não somente como "janelas" através das quais se veria um banco de dados de um modo particular, mas também como "cortinas" para esconder e proteger a informação, e como "telas" para interceptar qualquer operação de atualização que possa deixar a forma armazenada do banco de dados em um estado inaceitável.

PALAVRAS CHAVES:

Banco de dados, modelo relacional, operações de atualização, restrições, visões, álgebra de relações quocientes.

CONTENTS

1. UPDATES THROUGH VIEWS

The need for allowing various users to have different logical images of the structure of a data base has been widely recognized [GS,TK]. Such views or external schemas are supported by providing transformations of the stored data base representation into the representations for each user. If updates to the data base are expressed in terms of user views, however, the reverse transformations are also required. Because of the difficulty in specifying changes to the stored representation that are appropriate responses to updates on views, severe and rather arbitrary restrictions on which updates can be expressed through views have been proposed [Dat,CGT,ABC]. Generally, these restrictions permit updates only on views whose structures essentially coincide with the structure of some part of the stored representation. In this paper, we indicate how to permit a broad range of updates to be expressed through views while retaining the integrity of the stored representation.

The structure and operational policies of an enterprise, along with the division of responsibility among structural units for creating, using and changing information, determine the model of the enterprise that is represented in the data base. The structure of all information of relevance is expressed as the "conceptual schema", while portions of the information relevant to structural units are expressed as "external schemas" or user views [TK]. The organization of the information in each view is appropriate to the needs and responsibilities of the users of the view.

Certain limitations on activities involving the data base can be expressed as various kinds of constraints [Bro, CGT, HM, Sch, Sto, Web]. Here, we consider several types of constraints. Operational policies of the enterprise that constrain relationships among data base items can be formalized as policy constraints (e.g. "no employee's salary may be below the minimum wage", or "supervisors must have at least one year of experience in the enterprise"). Limitations on the information provided to or changeable by groups of users can be stated as authorization constraints. The consistency of redundant information in the data base can be expressed as consistency constraints (e.g. the enterprise budget should equal the sum of the departmental budgets). Finally, assumptions made by users about the relationships represented in their views are presented as assumption constraints. Policy, authorization, and consistency constraints are expressed in terms of the conceptual schema, while assumption constraints are expressed in terms

of views. Violation of a policy or an authorization constraint indicates a conflict between an action by some user and an enterprise policy. Violation of a consistency constraint indicates a failure to reflect activities of the enterprise in the data base correctly or completely. Assumption constraints provide a formal basis for assuring that users have compatible beliefs about the significance of the information represented in their view.

The mechanism for providing updates through user views plays a role in preserving various constraints. While views have primarily been considered as "windows" through which different users see the data base in different ways, we will also consider views as "shades" to limit what can be seen through a window, and as "screens" that limit how the data base can be affected through the window. Specifically, by properly constructing user views, it is possible to detect and intercept operations on the view that would lead to the violation of some types of constraints.

In this paper, we indicate, for one type of conceptual schema, how updates on user views can be transformed into updates on the conceptual schema. The criteria by which transformations are chosen are the following: (1) The change in a data base as seen through a view due to an update on the view should be exactly the change that would result if the stored representation coincided precisely with the view, (2) the change in the stored representation should not lead to the violation of any policy, authorization, or consistency constraint, and (3) the change to the data base as seen in other views should not lead to the violation of any assumption constraint. Thus, the updates allowed to one user are indirectly restricted by the assumption constraints of other users.

Although similar considerations arise for any data base organization, we will consider only the context in which both the stored data base representation and all views are expressed as sets of partitioned relations and the relations which compose the views are derived from the stored relations by operations of the algebra of quotient relation [FK]. Partitioned relations are relations whose tuples are grouped into blocks so that all tuples in a block have the same values for certain attributes, called the partitioning attributes. For example, if the relation EMPLOYEE (EMP#, JOBTITLE, MGR, SALARY) is partitioned on the set of attributes {JOBTITLE, MGR}

then each block will consist of the tuples for all employees with a given job title who work for a given manager.

The algebra of quotient relations includes six basic operations: (Let X and Y be sets of attributes and let R_x mean that relation R is partitioned on attributes in X).

(1) Partitioning

R_x / Y yields $R_{x \cup y}$

(2) De-partitioning

$R_x * Y$ yields R_{x-y}

(3) Projection

$R_x[Y]$ yields $R'_{x \cap y}$, with attributes Y

(4) Restriction

$R_x[A \ \Theta \ B]$

where Θ is a set comparison operator and A and B are attributes with Θ-comparable domains yields R'_x which is composed of a subset of the blocks of R_x. The blocks of R_x that are included in R'_x are those for which the sets of values in the block for attributes A and B respectively satisfy the set comparison operator Θ .

(5) Union

$R_x \oplus S_{x'}$

where X and X' are pair-wise compatible sets of attributes

[Cod2] yields $R'_{x''}$ whose attributes are pair-wise compatible with those of both R_x and $S_{x'}$. The attributes x" are those that correspond to x and x', and the blocks of $R'_{x''}$ each contain the union of the sets of tuples in the corresponding blocks of R_x and $S_{x'}$.

(6) Product (or Cartesian Product)

$$T_x \otimes S_y \text{ yields } R'_{x+y}$$

where the attributes of R'_{x+y} are the disjoint union of the attributes of R_x and S_y. The tuples of R'_{x+y} are all those that can be composed by concatenating a tuple of R_x with a tuple of S_y.

From these six basic operations, it is possible to synthesize join, division, intersection and difference operations. Thus, the algebra of quotient relations is "relationally-complete" [Cod2]. More precise definitions of these operations and examples of the utility of partitioned relations [FK] and a definition of partitioned relations as an abstract data type [Tom] are given elsewhere.

As is clear from the definitions above, the operation that differs most from its counterpart among conventional relational operators is restriction. An indication of the power of the restriction operator on partitioned relations is conveyed by the following two examples.

Example 1: Consider the relation MADE-OF(DEPT,PRODUCT,COMPONENT) in which each tuple specifies one component used by a department in assembling a product. The relation

$$(\text{MADE-OF}_{\{\text{DEPT}\}}[\text{PRODUCT} \supseteq \text{COMPONENT}])[\text{DEPT}]$$

then consists of precisely those departments which are self-sufficient in the sense that they make all the components used in any of their products. Because MADE-OF is partitioned on DEPT, the restriction treats all tuples for a given department at once. If any COMPONENT used by the department is not a PRODUCT of the department, then all tuples for that department are eliminated in the

restriction. The projection on DEPT simply picks out all departments corresponding to blocks of tuples in MADE-OF that satisfy the restriction.

Example 2: Consider the relation

EMPLOYEE(NAME,SALARY,MGR,MGR-SALARY).

Then the relation

$$\text{EMPLOYEE}_{\{\text{MGR}\}}[\text{SALARY} > \text{MGR-SALARY}]$$

contains all employees who work for managers that earn less than at least one of their employees. For the purpose of set comparison, A>B is interpretted as "some value for A is greater than some value for B". Three related operators, .>, >., and .>., express respectively, "all A > some B", "some A > all B", and "all A > all B". Similar quantified extensions of other comparison operators can also be used in restrictions.

It is important to distinguish the problem of supporting updates through views from several related but separable problems:

(1) Internal representation - once having specified insertions, deletions, or modifications to stored relations, we will not consider how these operations are carried out on the internal representation of the stored relations.

(2) Authorization - aside from enforcing the stated authorization constraints, our concern will be only with how to affect updates, not with assessing their permissibility.

(3) Concurrency - we will assume that a concurrency mechanism provides the effect of serializing update transactions. (Each transaction is a sequence of logically-connected updates which leaves the data base in a consistent state [Dat]).

In the next section, we describe a number of types of relationships that can hold between attributes,and that must be retained in user views.

Sections 3 and 4 discuss updates on user views that are derived by, respectively, single operations and combinations of operations. In section 5, we give examples of how the careful design of user views and the use of the transformations indicated in sections 3 and 4 can assure the preservation of constraints in the presence of updates through user views. Section 6 includes some comments on related topics and directions for further research.

2. TYPES OF RELATIONSHIPS IN VIEWS

In order to support updates through user views, the semantic significance or "meaning" of the information held in the data base must be considered. The definition of relations as sets of attributes fails to capture a range of important kinds of relationships among attributes . The fact that these properties must be retained no matter what relations are chosen to form the conceptual schema and user views provides guidance in the treatment of updates on user views.

Relationships may involve attributes from several relations (composed relationships), or may involve only some of the attributes of a single relation (contained relationships). For example, the pair of relations EMPLOYEE (NAME,DEPT) and DEPT-HEAD(DEPT,NAME) together represent a relationship between an employee and the head of his department. The last relationship is formed by the composition of the first two. Similarly, the relation SUPPLY (PROJECT,SUPPLIER,PART, QUANTITY) contains the relationship of projects to the sets of suppliers that supply some part to them.

Software engineering principles of modular design [YC] encourage the use of relational schemas in which each relation expresses a single fact. Such a schema has "high-strength" in that attributes within a relation are related and is "loosely-coupled" in that basic relationships are contained within single relations. Since normalization tends to produce such relations, conceptual schemas for relational data bases may consist of normalized relations [Codl, Fagin]. User views, however, will not in general consist of normalized relations, yet the relationships among attributes must be preserved. Since several distinct meaningful relationships (and arbitrarily many non-meaningful ones) can hold between two attributes, it is vital that each user know how the relationships in the stored relations are represented in his view. Assumption constraints can be used to check that one user's understanding of the relationships is compatible with the understandings of other users.

Because a data base typically models a changing world, relationships expressed in the data base are time-dependent. For example, the transfer of an employee to a new department changes the relationships of employee to department and employee to department-head. Time-dependent relationships may however,

have certain time-independent properties. For example, each employee may work in several departments at different times, but at any specified time, be associated with at most one department.

There are many significant properties of relationships that should be preserved in user views. Some properties presented elsewhere [Web] along with some additional ones are discussed below. For each property, we indicate the limitations imposed on relations that represent a relationship with that property.

(1) Functional dependence - If, at all times, each value of attribute A (or combination of values for attributes in set A) is related to at most one value of attribute B, then attribute B is functionally dependent on A[Dat].

For example, since each department has only one head at a time, the relationship between DEPT and HEAD-NAME is one in which HEAD-NAME is functionally dependent on DEPT. Thus, in the relation DEPT-HEAD (DEPT,HEAD-NAME), the functional dependence is violated if two tuples associate different HEAD-NAME's with the same department. Similarly, in the relation EMPLOYEE(NAME,DEPT, HEAD-NAME), each pair of tuples with the same DEPT must also have the same HEAD-NAME.

(2) Multi-valued dependence - If, at all times, each value of attribute A (or combination of values for attributes in set A) is associated with a set of values for attribute B (or a set of combinations of values for attributes in set B), then B is multi-valued dependent on A. (Note that functional dependence is the special case of multi-valued dependence in which each set of values for attribute B is a singleton set). For example, if employees are divided into teams, then EMP-NAME is multi-valued dependent on TEAM. Because the relation has only two attributes, it is impossible for any configuration of TEAM-ASMT(EMP-NAME,TEAM) to violate the multi-valued dependence. (Note, however, that if employees are members of at most one team, then the functional dependence of TEAM on EMP-NAME requires that each employee name appear only once in TEAM-ASMT).

If each team is assigned to several projects, then the relation PROJ-ASMT(EMP-NAME,TEAM,PROJECT) violates the multi-valued dependency of EMP-NAME on TEAM unless every set of tuples associating a specific team to some project consists of exactly the same set of employee names.

(3) Full-dependence - Both functional and multi-valued dependences of B on A are full-dependences if and only if B is not dependent on any subset of the set of attributes, A. Full dependences are of more interest than partial dependences since they correspond directly to semantic relationships.All dependences discussed in the rest of this paper will be full dependences unless specified otherwise.

(4) Totality - A relationship from A to B is total if the presence of an A value requires that it be associated with at least one B value. For example, if enterprise policy requires (policy constraint) that every employee be assigned to exactly one department at all times, then the relationship from EMP-NAME to DEPT is total. The totality is violated in the relation EMPLOYEE (EMP-NAME,DEPT) if any tuple associates an EMP-NAME with the value "undefined" (which is a special value distinct from any possible value of any attribute), or if some employees appearing under EMP-NAME in another relation are not included in this one. If the relationship from EMP-NAME to DEPT were not total, a tuple could associate an EMP-NAME with the undefined value in order to indicate the existence of an employee who is not assigned to any DEPT.

(5) Surjectivity - A relationship from A to B is surjective if the presence of a B value requires that it be associated with some A value. (Thus, surjectivity is the converse of totality). For example, since all department heads must have a department, the relationship from DEPT to HEAD-NAME is surjective. In the relation DEPT-HEAD(DEPT,HEAD-NAME), the surjectivity is violated if any tuple associates the value "undefined" for DEPT with some HEAD-NAME. Note that the primary key of a relation is a

set of attributes whose relationship to each other attribute in the relation is both a functional dependency and surjective.

(6) Containment - A relationship from A to B is a containment relationship (B is contained in A) if, at all times, the set of values for B is a subset of the set of values for A. For example, since department heads are also employees, the relationship between EMP-NAME and HEAD-NAME is one in which the set of HEAD-NAMEs is contained in the set of EMP-NAMEs. This containment property is violated in the relation EMPLOYEE(EMP-NAME,DEPT,HEAD-NAME) unless every name that appears as a HEAD-NAME in some tuple also appears in some tuple as an employee name.

(7) Permanence - A relationship from A to B is permanent if, once a value for A is associated with some B value, no other value of B is ever associated with that A value. For example, if employee numbers are never re-used even after an employee leaves the enterprise, then the relationship from EMP-NO to SOC-SEC-NO (social security number) is permanent. In the relation EMPLOYEE(EMP-NAME,SOC-SEC-NO,EMP-NO), the permanence is violated if any tuple associates a SOC-SEC-NO with an EMP-NO that is currently, or ever has been, associated with a different SOC-SEC-NO. (In this particular example, there is a one-to-one relationship between the two attributes, so the relationship from SOC-SEC-NO to EMP-NO is also permanent).

(8) Acyclic Composition - A relationship from A to B where attributes A and B have the same domain has the property of acyclic composition if and only if there is no sequence of pairs, $(a_1,b_1),(a_2,b_2),(a_3,b_3),\ldots,(a_n,b_n)$, such that $b_k = a_{k+1}$ for $k = 1,2,\ldots,n-1$ and $a_1 = b_n$. For example, if no employee can be his own manager, or his manager's manager, etc. then the relationship from EMP-NAME to MGR-NAME has the property of acyclic composition.

(9) Completeness - Rather than being a property of a relationship, completeness is a property of one attribute in the context

of a relation. If attribute A has a known,finite domain, then it is complete if and only if every value appears in the relation. For example, if the attribute DEPT in the relation DEPT-HEAD(DEPT,HEAD-NAME) is complete, then every department of the enterprise is represented by a tuple in the relation.

Consistency, policy and assumption constraints may require that some relationships have certain properties. As will be discussed in section 5, the preservation of these properties can be guaranteed by the proper selection of the set of user views.

3. UPDATES ON VIEWS DERIVED BY SINGLE OPERATORS

Consider a user view, V_k, of a set of stored relations, V_0. For sufficiently large k, we can specify a sequence of user views, $V_1, V_2, V_3, \ldots, V_{k-1}$, such that, for i from 1 to k, each relation in V_i is derived from relations in V_{i-1} with at most one operation of the algebra of quotient relations. In this section, we will consider which updates on view V_{k-1} are appropriate in response to updates on view V_k. We consider six cases to account for each possible operation by which V_k might have been derived from V_{k-1}.

Figure 1 (which is a generalization of Paolini's diagram [PP]) indicates how iterative application of the rules for handling updates on single operator views suffice to handle user views derived by any number of operations. The requested update, U_k, on user view V_k is transformed into an update U_{k-1} on view V_{k-1} according to the operation by which V_k was derived from V_{k-1}. The update on V_{k-1} is in turn transformed into an update on V_{k-2}, and so forth, until an update U_0, on V_0, the stored relations, is obtained. After U_0 is applied to V_0, to yield V_0', the updated view, V_k', can be derived. If the update transformation rules are correctly specified, V_k' will be precisely what the user would expect to result from applying U_k to V_k.

The updates for which we will specify transformations are insertion, deletion, and modification of a set of tuples. For insertion, the tuples must be enumerated, while for deletion and modification they may be either enumerated, or specified as the set of tuples that satisfy some condition. Any update that would cause some constraint to be violated must be intercepted, and the appropriate corrective action must be taken. For an authorization constraint, the update is rejected; for a consistency constraint, the inconsistency within the data base must be resolved; for a policy constraint, either the action represented by the update must be reversed or else the policy must be changed; for an assumption constraint, the discord between two users must be resolved by either rejecting the update of one or relaxing the assumptions of the other.

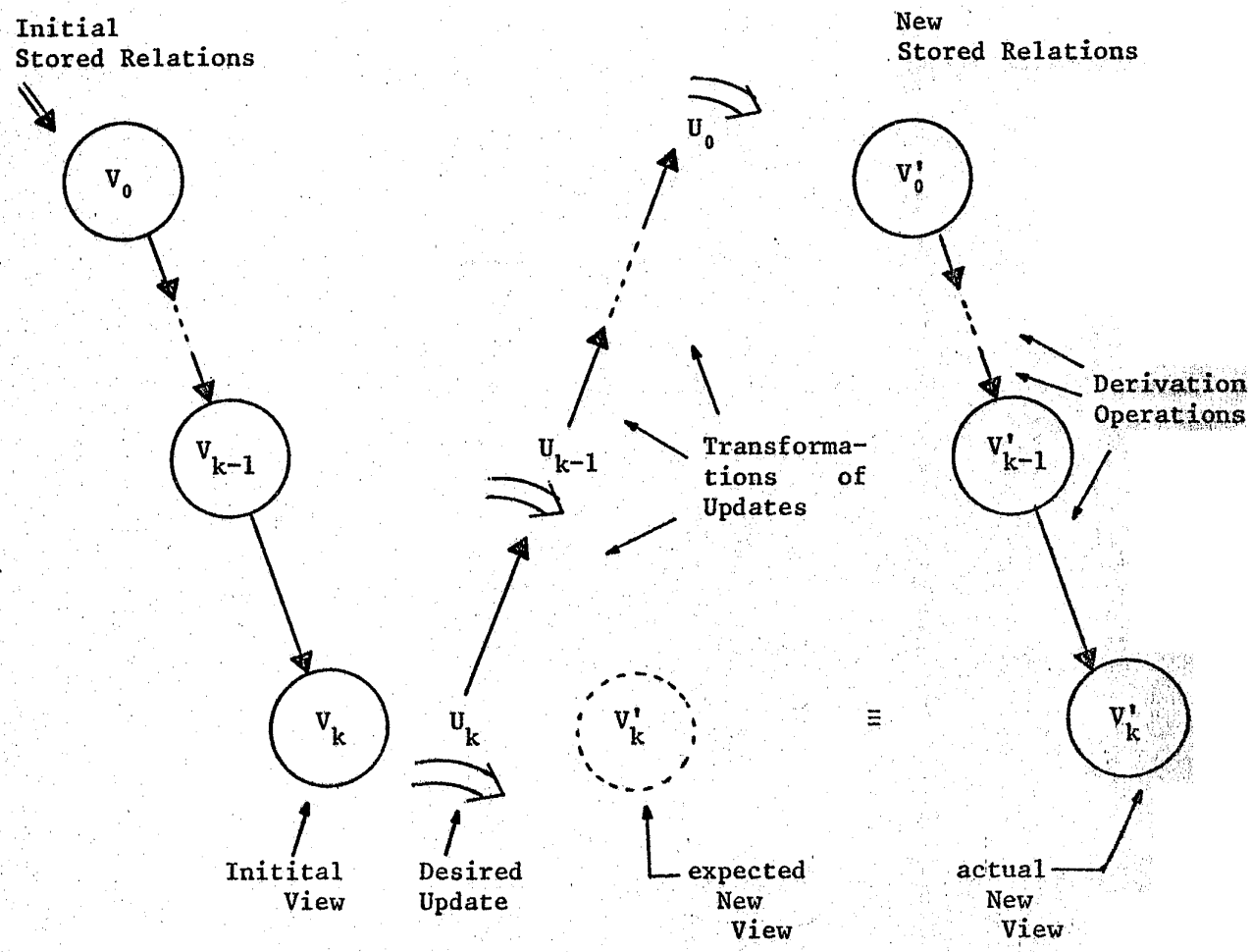


Figure 1. Iterative application of transformation rules for views derived by a sequence of operations.

An attempt to insert a tuple that is already present in a rela - tion is acceptable, but, of course, only one instance of the tuple remains in the relation after the operation. If the tuple being inserted has only some of its attribute values specified, it is sometimes difficult to determine whether or not it is a "new" tuple. Our convention will be that two tuples are the same if they (i) have the same defined value for at least one attribute, and (ii) do not have unequal defined values for any attribute. Thus, in the employee relation, EMP(NAME,DEPT,SALARY), the insertion of (SMITH, TOY, *) with (SMITH, *, 5000) already present would leave the single tuple (SMITH, TOY,5000) in the relation. The insertion of (*,TOY,5000) with (SMITH,*,5000) already present would similarly yield only (SMITH,TOY,5000). However, the latter situation is less clearly correct because the attribute for which the tuples have the same value is not a key of the relation. In most practical situations, it is possible and desirable to ensure that every tuple inserted into a relation have defined attribute values for some key of the relation.

In some cases, the transformation of an allowable update is uniquely determined, while in others, there may be several equally acceptable transfor- mations. In the latter situation, the choice of a specific transformation can be based on the meaning of the relations in each specific user view.

Table I summarizes the transformations of updates for views derived by a single operator and the limitations on when they are allowable. In the remainder of this section, we elaborate on the entries in the table by dis- cussing each operator in turn.

A. Partitioning and De-partitioning

Since insertion, deletion, and modification are all independent of partitioning, the operations expressed on R' can simply be applied to R .

B. Projection

The transformation of an insertion update on a projection requires that the "undefined" value (denoted here by "*") be placed in each tuple to be inserted for each attribute eliminated by the projection. For example, the relation EMPLOYEE(NAME,EMP# ,DEPT) might be viewed as EMP-NUMBERS(NAME,EMP#)= EMPLOYEE[NAME,EMP#] by the person who assigns employee numbers to new employees.

TABLE I. Transformation of updates on views derived with a single operation.

T - a set of tuples
B - a Boolean condition
A - an update algorithm
X,Y - sets of attributes
R' - a relation in V_k
R,S - relations V_{k-1}

Derived Relation R' \ Update Operation On R'	INSERT T INTO R'	DELETE TUPLES WITH B FROM R'	MODIFY TUPLES WITH B IN R' BY A
Partition R' = R/X	Apply to R	Apply to R	Apply to R
De-partition R' = R*X	Apply to R	Apply to R	Apply to R
Projection R' = R[X]	INSERT T* INTO R where T* is T augmented with "undefined" values for each attribute of R not in X.	Apply to R	Apply to R
Restriction R'= R[XΘY]	Apply to R	Apply to R	Apply to R
	(Disallow unless every affected block of R satisfies [XΘY] both before and after the update.)		
Union R'= R ⊕ S	Apply to R and S	Apply to R and S	Apply to R and S
Product R' = R ⊗ S	INSERT t_r INTO R INSERT t_s INTO S	DELETE t_r FROM R or DELETE t_s FROM S or both	MODIFY t_r IN R BY A_r or MODIFY t_s IN S BY A_s or both
	(where t_r and t_s are the projections of T on the attributes of R and S respectively, and Ar and As are update algorithms treating attributes of R and S respectively.)		
	(Disallow unless the cross-term condition is satisfied.)		

Then the update "INSERT <SMITH,17452> INTO EMP-NUMBERS" would be transformed into "INSERT <SMITH,17452,*> INTO EMPLOYEE ".

Deletions from and modifications to a projection can be directly applied to the relation which is projected. Note, however, that for each tuple deleted or modified in the projection, one or more tuples will be deleted or modified in the relation which is projected. For example, the relation SUPPLY(PROJECT,SUPPLIER,PART,QTY) may be viewed as

PROJ-SUPP(PROJECT,SUPPLIER) = SUPPLY[PROJECT,SUPPLIER]

by a person concerned only with the identities of the suppliers for each project. The termination of the association between the "Nova" project and supplier "United General Inc." would be indicated by

"DELETE TUPLES WITH
PROJECT=NOVA&SUPPLIER=UNITED-GENERAL FROM PROJ-SUPP"

which would be transformed by simply replacing PROJ-SUPP by SUPPLY. The transformed update would cause all tuples indicating that a part is supplied by United General to project Nova to be deleted from the SUPPLY relation, which is the appropriate effect. Similarly, consider modifying the relation PROJ-ASMT(EMP-NAME,TEAM,PROJECT) through the view TM-PROJ(TEAM,PROJECT) = PROJ-ASMT(EMP-NAME,TEAM,PROJECT). Changing the assignment of team "Ace" from project "Velha" to project "Nova" is indicated by

MODIFY TUPLES WITH
PROJECT=VELHA & TEAM=ACE IN TM-PROJ
BY PROJECT ← NOVA".

The transformation of this update simply replaces TM-PROJ with PROJ-ASMT, and the tuples of all members of the Ace team will be modified.

C. Restriction

Insertions, deletions, and modifications on R' are all transformed by simply replacing R' with R in the update operation. There is, however, a condition that must be satisfied if the update is to be allowed. The condition is that every block of R_y into which or from which a tuple moves as a result

of the update must satisfy the restriction both before and after the update. (By convention, an empty block always satisfies the restriction).If this condition did not hold, then the view of the user making the update would change unexpectedly by the appearance or disappearance of some tuples. Note that the acceptability of updates on views formed by restriction depends on the current data base contents, and thus can only be determined dynamically.

When deleting or modifying through a view formed by restriction, a full characterization of the tuples to be affected should be given to avoid inadvertently affecting others. This danger is avoided by requiring that deletions and modifications specify the value of all attributes on which the restricted relation is partitioned.

Figure 2a shows an instance of the relation SKILL-ASMT(EMPLOYEE, SKILL,PROJECT) which is partitioned by EMPLOYEE. We will consider updates on the three different views of SKILL-ASMT indicated in 2b. Assume that NOVA and BETA are priority projects, while VELHA is not. Then the views SOME-PRI,ALL-PRI, and ONLY-PRI are restrictions that include only those employees that respectively work on at least one priority project, work on both priority projects, and work on no projects other than priority ones. Figure 2c gives several example updates on the restrictions. The reader can verify that only the updates satisfying the condition stated above are allowable. Note that the bulk insertion and bulk deletion on ALL-PRI are allowable even though there is no allowable sequence in which the tuples can be inserted or deleted individually.

SKILL-ASMT (EMPLOYEE, SKILL, PROJECT)

SMITH	PLUMBER	VELHA
ARNOLD	ELECTRICIAN	VELHA
ARNOLD	CARPENTER	NOVA
ARNOLD	CARPENTER	BETA
JONES	PLUMBER	BETA
JONES	PLUMBER	VELHA
WILSON	ELECTRICIAN	NOVA

(a) The relation SKILL-ASMT partitioned on EMPLOYEE

SOME-PRI = SKILL-ASMT [PROJECT $\approx$ {NOVA,BETA}]

(where X $\approx$ Y iff X $\cap$ Y $\neq \emptyset$)

ARNOLD	ELECTRICIAN	VELHA
ARNOLD	CARPENTER	NOVA
ARNOLD	CARPENTER	BETA
JONES	PLUMBER	BETA
JONES	PLUMBER	VELHA
WILSON	ELECTRICIAN	NOVA

ALL-PRI = SKILL-ASMT [PROJECT $\supseteq$ {NOVA,BETA}]

ARNOLD	ELECTRICIAN	VELHA
ARNOLD	CARPENTER	NOVA
ARNOLD	CARPENTER	BETA

ONLY-PRI = SKILL-ASMT [PROJECT $\subseteq$ {NOVA,BETA}]

WILSON	ELECTRICIAN	NOVA

(b) Three restrictions of SKILL-ASMT

update	allowable?	why not?
On SOME-PRI:		
INSERT (ADAMS,PLUMBER,NOVA)	yes	
INSERT (ADAMS,PLUMBER,VELHA)	no	inserted tuple doesn't appear
INSERT (SMITH,PLUMBER,NOVA)	no	an extra tuple also appears
DELETE (WILSON,ELECTRICIAN,NOVA)	yes	
DELETE (SMITH,PLUMBER,VELHA)	no	tuple is not in view
On ALL-PRI:		
DELETE (ARNOLD,ELECTRICIAN,VELHA)	yes	
DELETE (ARNOLD,CARPENTER,NOVA)	no	entire block disappears
DELETE {(ARNOLD,CARPENTER,NOVA) (ARNOLD,CARPENTER,BETA) (ARNOLD,ELECTRICIAN,VELHA)}	yes	

INSERT (SMITH,PLUMBER,NOVA)	no	tuple doesn't appear
INSERT {(SMITH,PLUMBER,NOVA) (SMITH,PLUMBER,BETA) (SMITH,PLUMBER,VELHA)}	yes	

On ONLY-PRI:

INSERT (WILSON,ELECTRICIAN,BETA)	yes	
INSERT (WILSON,ELECTRICIAN,VELHA)	no	entire block disappears

(c) Examples of allowable and not allowable updates.

Figure 2. Examples of updates on views derived by restriction.

As an example of the necessity of fully characterizing the tuples to be affected, consider the following. In order to release Arnold from any duties as an electrician or to move him from the Velha project to the Beta project, a user of the ALL-PRI view might use updates

DELETE TUPLES WITH SKILL=ELECTRICIAN FROM ALL-PRI

or

MODIFY TUPLES WITH SKILL=ELECTRICIAN IN ALL-PRI
BY PROJECT ← BETA

However, because Arnold's name (the partitioning attribute) is not specified in the updates, Wilson as well as Arnold would be affected by the transformed updates applied to SKILL-ASMT.

D. Union

Insertions, deletions and modifications on a view which is a union should be applied to each of the relations merged by the union operation. The only exception is that insertion might be done to only one relation or the other if some basis for the choice is available. Similarly, if it is known somehow that no tuple affected by a deletion or modification lies in one of the relations contributing to the union, then the update need not be applied to that relation. Because of the choice in inserting in a union, a modification may not have the same effect on the stored relations as the apparently

equivalent deletion and insertion.

E. Product

Updates to views formed by a product are allowable only under a stringent condition that, in effect, never holds. In the next section, we indicate a way of avoiding this apparently severe limitation.

Let T be a set of tuples to be inserted into $R' = R \otimes S$ and let t_r and t_s be the projections of T on the attributes of R and S, respectively. Then T is inserted into R' by inserting t_r into R and t_s into S. But

$$(R \oplus T_r) \otimes (S \oplus t_s) = (R \otimes S) \oplus (R \otimes t_s) \oplus (t_r \otimes S) \oplus (t_r \otimes t_s),$$

so the insertion is allowable (i.e. the intended effect on the view is achieved) if and only if

$$(R \otimes S) \oplus T = (R \otimes S) \oplus (R \otimes t_s) \oplus (t_r \otimes S) + (t_r \otimes t_s).$$

A sufficient condition for this equality to hold is

$$[(T = t_r \otimes t_s) \text{ and } (R = t_r \text{ or } S = t_s \text{ or } R=S=\emptyset)]$$

which we will refer to as the "cross-term condition". (Appendix A contains a development of the condition). The first part of the cross-term condition requires that T consist of all possible pairings of tuples from its projection on R attributes with tuples from its projection on S attributes. This requirement is trivially satisfied if T consists of only one tuple. The second part of the cross-term condition requires that either R or S be identical to the projection of T on the appropriate attributes, or else that both R and S be empty.

While the cross-term condition is not "necessary" (in the formal sense) for an update to be allowable, only intricately contrived examples in which the set of tuples, T, has a carefully chosen structure yield allowable insertions in the absence of the cross-term condition. For deletion and modi-

fication, the same cross-term condition is sufficient to guarantee that the update is allowable (but note that $R=S=\emptyset$ cannot hold). Deletions are accomplished by deleting from either R or S or both and modifications by changing either some tuples of R or some tuples of S or both.

4. VIEWS DERIVED WITH SPECIFIC OPERATOR COMBINATIONS

In the previous section, we discussed the transformation of update operations on views formed with each of the six basic operations of the algebra of quotient relations. We indicated that views derived using more than one operator could be treated by iterative application of the transformation rules for single operations. In this section, however, we will give special attention to certain combinations of operations for which some updates are allowable that would not be allowable if the transformations for the operators were applied individually. Some combinations of operations are particularly important because they synthesize more powerful operations (such as "join") from the six basic operations.

A. Restriction of a Fully-Partitioned Relation

If R is fully-partitioned (i.e. partitioned on all attributes, so that each tuple is a block), then the restriction operation $R' = R[A\Theta B]$ specializes to the usual restriction operation on relations [Dat]. Since each tuple is a block in R, update operations on R' cannot have the side-effects of making tuples appear or disappear in R as a result of occupying the same block of R as a tuple affected by the update. All deletions and all insertions or modifications for which the inserted or modified tuples satisfy the restriction are allowable. The allowability of updates no longer depends on the current data base contents.

B. Projection of a Restriction

In some views, a restriction is immediately followed by a projection that eliminates the attribute on which the restriction was based. Stonebraker [Sto] gives an example which shows that the restriction may determine a specific value for one or more attributes eliminated by the projection. Thus, in transforming an insertion into a view formed by restriction followed by projection, we can do better than supplying "undefined" values for all the attributes not in the projection. For example, if the relation TEAM-ASMT(EMP-NAME,TEAM) associates employees with teams and each team leader is limited (authorization constraint) to adding employees only to his own team, then an appropriate view for the leader of team Bravo to have is

BRAVO-TM (EMP-NAME) = (TEAM-ASMT [TEAM=BRAVO]) [EMP-NAME].

Then insertion of the tuple (WILSON) into BRAVO-TM would be transformed into the insertion of (WILSON,BRAVO) rather than (WILSON,*) into TEAM-ASMT. Similarly, if a restriction demands equality between two attributes, only one of which survives the projection, then the value of the surviving attribute is duplicated for the other attribute when an insertion into the projection is transformed into an insertion into the relation that is restricted and projected.

C. Union of Restrictions

If a view is formed as the union of several non-overlapping restrictions on the same relation, then the choice of the component relation into which a tuple inserted into the union must be placed is determined. Consider again the relation TEAM-ASMT and its restrictions by teams,

BRAVO-TM = TEAM-ASMT[TEAM=BRAVO]

and so forth for all other teams. An insertion of (WILSON,BRAVO) into the view

TM-SET = BRAVO-TM ⊕ ACE-TM ⊕ ...

should clearly be transformed into an insertion into only BRAVO-TM.

D. Union of Partitioned Relations

A weaker mechanism (heuristic only) for resolving the ambiguity of transforming insertions into unions is available if the union merges partitioned relations. If only one of the relations merged by the union has a non-empty block into which the tuple being inserted could be placed, then we may choose to transform the insertion into the union into an insertion only into that relation, with the intention of grouping tuples with the same values for the partitioning attributes.

E. Restriction of a Product

Restrictions of products are particularly important in the algebra of quotient relations since they are used to synthesize "join" operations. If

attributes A and B are comparable and they partition respectively relations R and S, then $R' = (R_A \otimes S_B)$ [A=B] is the "equi-join" [Dat] of relations R and S. Despite the severe limitations seen in the last section on updates to products, updates to restrictions of products are allowable in many situations of practical importance.

In the case of a natural join, the restriction eliminates from the product all blocks except those formed from corresponding pairs of blocks from the relations contributing to the product. Thus, by successively considering the interaction of each pair of corresponding blocks, we consider all the interactions between the contributing relations.

The condition for updates to a natural join (formed as the restriction of a product) to be allowable is that each pair of corresponding blocks satisfy the cross-term condition of section 3, with respect to the tuples entering or leaving the block due to the update. The cross term condition is guaranteed to hold if either

(i) the relationship from A to B is surjective and all other attributes of R are functionally dependent of A,

or

(ii) the relationship from A to B is total and all other attributes of S are functionally dependent on B.

(This is equivalent to the fact a "loss-less join" is guaranteed if the join attribute(s) is (are) a key of at least one of the relations [RD]). For example, consider the relation BOSS(EMP-NAME,DEPT,HEAD-NAME) which is the join on DEPT of EMPLOYEE(EMP-NAME,DEPT) and DEPT-HEAD(DEPT,HEAD-NAME). (We omit the redundant occurrence of the DEPT attribute in the BOSS relation). The insertion of the tuple (ARNOLD,ACCOUNTING,BLAKE) into the BOSS relation could have any one of several effects. If some tuple of BOSS already indicates that BLAKE is head of the accounting department, then the insertion is transformed into just INSERT (ARNOLD,ACCOUNTING) INTO EMPLOYEE. If no tuple of BOSS refers to the accounting department, then two insertions are needed: INSERT(ACCOUNTING,BLAKE) INTO DEPT-HEAD and INSERT (ARNOLD,ACCOUNTING) INTO EMPLOYEE.
If either Blake or Arnold is associated with some department besides accounting, then the insertion is intercepted because it would lead to a violation of the functional dependency of either HEAD-NAME on DEPT or DEPT on EMP-NAME.

Deletion from a view formed by the restriction of a product is allowable under the same conditions as is an insertion. The transformation of a deletion typically results in a deletion in only one of the relations being joined. A tuple in either of the relations being joined can only be deleted if all tuples to which it contributes in the join are among those being deleted from the join.

Modifications to a join may have undesirable side effects. Stonebraker [Sto] gives an example involving the join of relations relating employees to departments and departments to floors. The attempt to move just one employee from one floor to another by modifying one tuple of the join is allowed by Stonebraker to move also all other employees in the same department. We would intercept such an update as an attempt to violate a functional dependency (assuming that all members of one department work on the same floor). In order to move an entire department from one floor to another, the modification to the join would have to include all employees in the department explicitly.

F. Projection of a Restriction of a Product

Join operations (restrictions of products) are often followed by projections which eliminate either one or both of the join attributes. For example, the join of TEAM-ASMT(EMP-NAME,TEAM) and TM-PROJ(TEAM,PROJECT) might be viewed as PROJ-ASMT(EMP-NAME,TEAM,PROJECT) or even EMP-PROJ(EMP-NAME, PROJECT) by projecting out one or both occurrences of the TEAM attribute. An insertion of (CLARK,ALPHA,NOVA) into PROJ-ASMT can clearly lead to the duplication of ALPHA before insertion to the product, since the restriction eliminates any tuples of the product with differing team names. However, the insertion of (CLARK,NOVA) must be disallowed since there is no way of determining the team. (Use of the "undefined" value in this situation is undesirable since it would appear in two different places, and there is no way to indicate that the same unspecified value must appear in each place). The attempt to delete (WILSON,VELHA) must also be disallowed since there is no way of knowing whether to remove Wilson from teams associated with project Velha, or to deassign all Wilson's teams from project Velha. It is reasonable to require that such a deletion be accomplished by updates from users with the views TEAM-ASMT or TM-PROJ.

G. Union of a Join and a Remainder

In the last example of the previous section, the fact that PROJ-ASMT was the join of two relations caused TEAM to be multi-valued dependent on EMP-NAME in PROJ-ASMT. If, however, the assignment of a team to a project does not necessarily include all members of the team, then the multi-valued dependency would no longer hold, and PROJ-ASMT would no longer be decomposable into two component relations. In this situation, a different form of decomposition, first suggested by Furtado and Kerschberg [FK], might be useful. Just as we may express an integer, i, which is not divisible by j, in the form

$$i = j * q + r$$

where q and r are respectively the quotient and remainder of dividing i by j, we may also decompose relations that do not involve multi-valued dependencies into the join of two relations plus some additional tuples. While natural motivations for such a representation are difficult to identify, the representation may be of importance as an artificial means of easing the limitations on insertions into joins. Updates not allowed on a join may be allowable on the union of a join and a remainder relation whose attributes are the same as those of the join (after the duplicate copy of the join attribute is projected out).

For example, assume that projects include tasks and each task requires a set of skills. If each task requires the same set of skills in every project where it occurs, then skills are multi-valued dependent on tasks, and the relation PTS(PROJECT,TASK,SKILL) can be decomposed into PT(PROJECT,TASK) and TS(TASK,SKILL). The relation PTS can be recovered as the natural join of PT and TS on TASK.

If, however, some tasks occasionally require additional skills in certain projects, then PTS can no longer be represented as simply the join of PT and TS. Instead, it is necessary to form another relation specifically to indicate the occasional additional skills needed for certain tasks on certain projects. Then PTS is decomposable into PT,TS and OTHER-PTS, and is recovered as the union of OTHER-PTS with the join of PT and TS. Figure 3 shows how updates on such a representation work. Starting with the design task requiring the same skills in each project, we wish to indicate that, on project Nova only, the design task now also requires the radiation-control skill. If PTS were simply a join, this update could not be achieved, but with the remainder re-

presentation, the new tuple is simply placed in OTHER-PTS (insertion # 1). If a new project, Beta, involving design with its usual skills, is defined, then an insertion into PT suffices (insertion # 2). If it is determined that design always requires some previously omitted skill, then an insertion into TS is needed (insertion # 3). The last case of figure 3 (insertion # 4) is more complex. It could be achieved by simply placing the indicated tuples into OTHER-PTS. It is desirable, however, to maintain a "reduced form" in which OTHER-PTS contains as few tuples as possible. Algorithmically maintaining such a reduced form through updates appears to be difficult.

The similar idea of a "deduction relation", which contains tuples that do not appear in the view but are generated by the join, facilitates representation of situations where the exceptional cases are missing tuples rather than extra ones. Remainder relations and deduction relations can be used simultaneously, but the problem of maintaining reduced form is made more difficult.

Providing the view

PTS(PROJECT,TASK,SKILL)

from the relations

PT(PROJECT,TASK), TS(TASK,SKILL), and
OTHER-PTS(PROJECT,TASK,SKILL)

as

PTS = (PT ⊗ TS) [TASK=TASK] ⊕ OTHER-PTS.

(The relations PTS,PT,TS, and OTHER-PTS are all partitioned on TASK, and we show only the block for TASK = DESIGN below).

Initial PTS:

PROJECT	TASK	SKILL
NOVA	DESIGN	STRESS-ANALYSIS
NOVA	DESIGN	HUMAN-FACTORS
VELHA	DESIGN	STRESS-ANALYSIS
VELHA	DESIGN	HUMAN-FACTORS

Initial PT:

PROJECT	TASK
NOVA	DESIGN
VELHA	DESIGN

Initial TS:

TASK	SKILL
DESIGN	STRESS-ANALYSIS
DESIGN	HUMAN-FACTORS

(Initial OTHER-PTS is empty)

1. INSERT (NOVA,DESIGN,RADIATION-CONTROL) INTO PTS

   ⇒ Apply the insertion to OTHER-PTS

2. INSERT {(BETA,DESIGN,STRESS-ANALYSIS)
   (BETA,DESIGN,HUMAN-FACTORS)} INTO PTS

   ⇒ INSERT (BETA,DESIGN) INTO PT

3. INSERT {(NOVA,DESIGN,FORECASTING)
   (VELHA,DESIGN,FORECASTING)
   (BETA,DESIGN,FORECASTING)} , INTO PTS

   ⇒ INSERT (DESIGN,FORECASTING) INTO TS

4. INSERT {(VELHA,DESIGN,RADIATION-CONTROL),
   (BETA,DESIGN,RADIATION-CONTROL)}

   ⇒ INSERT (DESIGN,RADIATION-CONTROL) INTO TS
   DELETE (NOVA,DESIGN,RADIATION-CONTROL) FROM OTHER-PTS

Figure 3. Some operations on a view composed of the union of a join and a remainder.

5. ENFORCING AND CHECKING CONSTRAINTS BY VIEWS

In the last two sections, we have presented transformations of updates on views into updates on the relations from which the view is derived. In some cases, the relationships between attributes were required to have certain properties for the updates to be allowed. In this section, we demonstrate through a sequence of examples how such properties, as well as other constraints, can be either automatically enforced or at least checked using carefully designed user views.

Enforcing constraints, involves using views as screens, to limit the changes that each user can make in the data base. The data base is initialized to a state that satisfies all constraints, then updates are permitted only through user views that are guaranteed to intercept any update that could possibly cause the data base to enter a state that violates some constraint.

Checking constraints, on the other hand, involves constructing new views to serve as windows. In order to check some constraint through a view, we derive a second view from the given one, removing by the derivation any violations of the constraint in the first view. If the newly derived view is not identical to the initial view, then the initial view did not satisfy the constraint. In some situations, an attempted update must be provisionally executed, and then un-done if checking reveals that some constraint has been violated.

Enforcing is an aggressive approach to preserving constraints, whereas checking is a defensive approach. Since checking is difficult to carry out efficiently, it is preferable to enforce constraints whenever possible. The allowability of some updates can be determined in terms of the update, the view on which the update is expressed, and the view to which the transformed update is to apply. For other updates the current contents of the data base must be examined in order to determine whether or not the update is allowable. The former type of updates can be handled by enforcement of constraints alone, while the latter require that constraints be checked.

Most authorization constraints can be enforced by appropriately designed user views. For example, the relation EMPLOYEE(EMP-NAME,JOB-TITLE,

SALARY) might be seen through a view in which the salary is projected out, or in which a restriction eliminates all tuples with, for example, the job title "industrial spy". In this way, not only is the sensitive information itself concealed, but even its existence can not be detected by users of the view. Also, users are unable to modify or delete the sensitive information since the variable names are undefined with respect to their view.

More complex authorization constraints can also be handled. If a user is allowed to see all salaries, but may not increase any salaries beyond $ 50,000, then he can be given a direct view of EMPLOYEE through which no updates are allowed plus a view that allows updates but restricts salaries to be $ 50,000 or less. Consider, however, an authorization constraint which allows raises to be given to only those employees who currently earn less than the average salary in their department. Since the permissibility of a raise to an individual can only by judged by examining the current data base contents, no view is able to enforce the constraint; it can only be checked.

Consistency constraints, by their nature, involve at least two pieces of information. For this reason they cannot be enforced or even checked by views alone. Remember, also, that consistency constraints are not guaranteed to hold after every update, but only at the end of each transaction. Thus, updates that introduce temporary inconsistencies during the execution of a transaction must be permitted. For example, the consistency constraint that the enterprise budget equal the sum of the departmental budgets is tempora - rily violated during each transaction that changes any departmental budget , because either the change to the departmental budget or the change to the enterprise budget is done before the other.

Some policy constraints can be enforced by views. If policy dictates that all salaries must satisfy the minimum wage, then every view through which salaries are established or changed should include the restriction, SALARY $\geq$ MIN-WAGE. Similarly, if every employee must be associated with some department as a matter of policy (the relationship from employee to department is total), then insertion of tuples representing employees should be permitted only through views that include the restriction $\tilde{[}$DEPT $\neq$ "*"$\tilde{]}$, which permits the insertion of employees only if their department is specified.

Because such properties of relationship as completeness, functional dependence, and multi-valued dependence are based on values in tuples other than those directly affected by an update, the current data base contents is necessarily involved in checking that such constraints are preserved through an update. For example, the constraint that no one can be associated with more than one project at a time requires that the relationship from employee name to project be a functional dependency. If the association of employees to projects is a contained relationship in a relation that includes name, project and task for each task of the employee, then the functional dependency from employee name to project can be checked by the derivation expression

$$\text{EMP-VW} = (((\text{EMP}_{\{NM,PJ\}} \otimes \text{EMP}_{\{NM\}})[NM_1=NM_2])[PJ_1=PJ_2])[NM_1,PJ_1,TK_1].$$

In this expression, two instances of EMP relation with different partitionings are joined on name. The original partitionings guarantee that one project attribute of the join can have only a single value in each block of the join, but that the other project attribute will have more than one value if and only if tuples indicate that the employee is associated with more than one project. Thus, the restriction that the sets of values for the two project attributes be identical deletes the blocks corresponding to employees associated with more than one project. The final projection simply causes EMP-VW to have the same attribute structure as EMP. An attempt to enter into EMP-VW a tuple that associates an employee with a department different than the one with which he is already associated would instead cause removal of the employee from EMP-VW. Because this behavior of the view is not consistent with the intention of the update, the update is unacceptable, and EMP is restored to its state before the attempted update.

The derivation expression for PROJ-VW is complex and does not necessarily represent the procedural method of checking the functional dependency. It does, however, provide the formal basis for assuring the correctness of the interactions between the process of supporting updates through user views and the process of checking constraints, whether by actually deriving views or by more direct methods.

Because the acceptability of some updates is guaranteed by the presence of certain properties (as seen in sections 3 and 4), the effort in assessing the permissibility of some updates can be reduced by noting the pre-

sence of constraints that preserve the properties.

Constraints that require the preservation of multi-valued dependencies can be checked in a manner similar to that used for functional dependence. Assume that the relationship of team to employee name is a multi-valued dependency, and that employees can belong to more than one team. Then the view

PROJ(NM,TM,PJ)

must contain tuples for the same set of employees for each project to which that team is assigned. This requirement can be checked by doing all insertions, deletions and modifications through the view

$$\text{PROJ-VW} = (((\text{PROJ}_{\{\text{TM,PJ}\}} \circledast \text{PROJ}_{\{\text{TM}\}})[\text{TM}_1=\text{TM}_2])[\text{NM}_1=\text{NM}_2])[\text{NM}_1,\text{TM}_1,\text{PJ}_1].$$

This derivation has the effect of eliminating all tuples associating a team and a project unless there is one tuple for every employee that is assigned to any project in connection with that team.

If a policy constraint requires that employees only be assigned to projects with leaders, where the view for inserting employees includes the relations EMP(NM,PJ) and PROJ(PJ,D), then the constraint can be checked by doing updates to EMP on EMP-VW instead where

$$\text{EMP-VW} = (((\text{EMP}_{\{\text{NM}\}} \circledast \text{PROJ}_{\{\text{PJ}\}})[\text{PJ}_1=\text{PJ}_2])[\text{LD} \neq \text{"*"}])[\text{NM,PJ}].$$

Since the derivation eliminates any employee assigned to a project that is either not in PROJ or has an undefined leader in PROJ, insertions of employees associated with leader-less projects are impossible.

Assumption constraints can be preserved in at least two different ways. The most staightforward is to incorporate checks of the constraints into the derivation of the view. In this case, the view must be re-derived whenever any user attempts an update that might jepordize the constraint. The necessity for re-deriving the views of several users each time any user makes an update is clearly undesirable.

A second approach to maintaining assumption constraints is to determine corresponding constraints on the stored relations such that the preservation of these constraints guarantees the preservation of the assumption constraints. The new constraints on the stored relations can then be treated in exactly the same fashion as policy or consistency constraints. That is, for some constraints, appropriately constructed views for all updates can enforce the constraint, while for others, the constraint must be checked by re-derivation of the view after each update.

The amount of checking required to preserve constraints increases with the number of different users allowed to change the information involved in the constraint. For this reason, there is a temptation to limit the number of users permitted to update information. Unfortunately, such an attempt to decrease the expense of checking constraints will increase the amount of coordination required to occur among the users. For example, if only the Personnel department is authorized to update information about employees, then all other users who initiate changes affecting employees must do so by requesting that the change be made by the Personnel department. While such centralized control has been desirable in the past, it is no longer necessary if appropriate controls on access to and modification of information are provided by the provision of user views for updates.

Elsewhere [SF], we give an extended example of how several users cooperate in a simplified enterprise model, and how the careful design of views for updates can guarantee the preservation of constraints.

6. DISCUSSION AND CONCLUSIONS

We have treated each derived view as consisting of relations derived by a few operations from other relations, which may be stored relations, or may themselves be derived relations. Thus, it is possible to express one view in terms of others, as has been claimed desirable by some [TK]. Our approach also provides views with a greater degree of independence from the conceptual schema than do previous approaches. While the conceptual schema should not change often, users should be insulated as much as possible from even occasional changes [Dat].

As an example of the independence of views from the conceptual schema, consider the attributes employee name, skills, and project, where each employee is associated with only one project (i.e. the relationship from name to project is a functional dependency). The conceptual schema might include either the relation

ESP (NAME,SKILL,PROJECT)

or the two relations

ES (NAME,SKILL) and EP (NAME,PROJECT).

These relations are related as follows:

$$ES = ESP\ [NAME,SKILL]$$
$$EP = ESP\ [NAME,PROJECT]$$
$$ESP = ((ES \otimes EP)\ [NAME_1=NAME_2])\ [NAME_1,SKILL,PROJECT]$$

Because ESP can be expressed in terms of ES and EP, a change from the choice of ES and EP for the conceptual schema to the choice of ESP may affect the user views only in that each occurrence of ES or EP in a derivation expression is replaced by the equivalent expression involving ESP. The change from ESP to ES and EP can be handled similarly.

No matter what choice of conceptual schema is made, any constraints (like the functional dependency from name to project) must be preserved. Because the constraints are preserved in both choices of conceptual schema, the effect of any update as observed through a view is also independent of the choice of conceptual schema.

The preservation of constraints using specially-designed user views, as we have discussed, contributes to the semantic characterization of relational data bases, and relates to several other recently developed concepts. Roles [BD] allow a single entity to be represented in several ways in the data base simultaneously. For example, the same person may be both an employee and a shareholder of a company. Since the identifiers of employees (employee numbers) and shareholders (shareholder numbers) are typically not related, it is difficult to assemble all the information about an individual. We would handle the concept of roles in our approach by using entity relations as well as constraints. Entity relations provide a cross-reference table for each type entity to show the correspondence among the different identifiers that refer to the same entity. Constraints assure, where necessary, that update transactions account for the fact that an update of information about an entity in one of its roles may necessitate a corresponding update to information about the entity in other roles.

"Generalizations" [SS] accommodate sets of entities in which all entities have certain attributes, but various identifiable subsets of entities differ in the additional attributes they possess. For example, a company has employees with many different job titles. While certain attributes apply to all employees (name, salary, years with company, etc.), others are specific to each job title (sales quota for salesmen, typing speed for secretaries, etc.). Because the relational model requires that all tuples in one relation have the same attributes, the relation describing employees would have to have many attributes, of which most are irrelevant with respect to each specific employee. The few attributes with meaningful values for an employee would depend on the employee's job title. A more sophisticated solution would consist of one relation containing the attributes relevant to all employees plus one relation for each job title, containing the attributes relevant to the job title. In this case, assembling all the information about each employee would involve a "join" (on employee number and job title) of the relation containing the common attributes with the (heterogeneous) set of relations for the job titles, yielding another heterogenous set of relations. Such a heterogenous set of relations may be thought of as a single partitioned relation of generalized form in that different blocks may have different attribute structures.

In this paper, we have demonstrated that a broad range of updates can be safely expressed through views. We have indicated the kinds of transformation techniques required to determine the operations on the stored rela-

tions that are appropriate to affect an update expressed through a view. The key concept in devising the transformations was that certain properties of relationships, which can be indicated as constraints, are present in every correct view.

With carefully designed derivation expressions for views, it is possible to guarantee the preservation of certain declared constraints. While the derivation expressions become unwieldy and difficult to understand, various optimizations can make their use more efficient than is apparent. But even if the use of such derivation expressions is expensive, we feel that it is preferable to the alternative: to allow updates to be expressed only through views whose structures happen to coincide with some portion of the stored relations.

REFERENCES

ABC Astrahan, M.M. et al, System R: A Relational approach to data base management, TODS 1,2 (June 1976), 97-137.

BD Bachman, C.W. and M. Daya, The Role concept in data models, Proc. of Very Large Data Bases Conference (1977).

Bro Brodie, M.L., A Formal approach to the specification and verification of semantic integrity in data bases, Ph.D. Thesis, University of Toronto, in preparation.

CGT Chamberlin, D.D., J.N. Gray, and D.D. Traiger, Views, authorization and locking in a relational database system, Proc. AFIPS NCC, vol.44 (1975).

Cod1 Codd, E.F., Recent investigation into relational data base systems, Proc. IFIP Congress, North-Holland Publ. Co., Amsterdam (1974).

Cod2 Codd, E.F., Relational completeness of data base sublanguages, in Data Base Systems, ed. R. Rustin, Prentice-Hall, Toronto (1972).

Dat Date, C.J., An Introduction to Database Systems, 2nd edition, Addison-Wesley, London (1977).

EC Eswaran, K.P. and D.D. Chamberlin, Functional specifications of a subsystem for data base integrity, Proc. of Very Large Data Bases Conference (1975).

FK Furtado, A.L. and L. Kerschberg, An Algebra of quotient relations, Proc. SIGMOD Conference (1977).

GLP Gray, J.N., R.A. Lorie, and G.R. Putzolu, Granularity of locks in a large shared data base, Proc. of Very Large Data Bases Conference (1975).

GS GUIDE/SHARE Data Base Task Force, Data base management system requirements (1971).

HM Hammer, M.M. and D.J. McLeod, Semantic integrity in a relational data base system, Proc. of Very Large Data Bases Conference (1975).

PP Paolini, P. and G. Pelagatti, Formal definition of mappings in a data base, Proc. SIGMOD Conference (1977).

RD Rissanen, J. and C. Delobel, Decomposition of files, a basis for data storage and retrieval, IBM Research Report RJ 1220 (1973).

SC Smith, J.M. and P.Y.T. Chang, Optimizing the performance of a relational database interface, CACM 18,10 (1975).

Sch Schaefer, M., On certain security issues relating to the management of data, in The ANSI/SPARC DBMS Model, ed. D.A. Jardine, North-Holland Publ. Co., Amsterdam (1977).

SF Sevcik, K.C. and A.L. Furtado, Complete and compatible sets of update operations, Technical Report, Pontificia Universidade Católica (1977).

SS Smith, J.M. and D.C.P. Smith, Database abstractions: aggregation and generalization, TODS 2,2 (1977).

Sto Stonebraker, M., Implementation of integrity constraints and views by query modification, Proc. SIGMOD Conference (1975).

TK Tsichritzis, D. and A.Klug, The ANSI/X3/SPARC DBMS framework, Technical Note 12, Computer Systems Research Group, University of Toronto (1977).

Tom Tompa, F.W., A Practical example of the specification of abstract data types, Technical Report, Pontificia Universidade Catolica (to appear)

Web Weber, H., A Semantic Model of integrity constraints on a relational data base, in Modelling in Data Base Management Systems, ed. G.M. Nijssen, North Holland Publ. Co., Amsterdam (1976).

YC Yourdon, E. and L.L. Constantine, Structured Design, Yourdon Inc.(1975).

ACKNOWLEDGMENTS

We are grateful to F.W. Tompa for many helpful comments, criticisms and suggestions, and to J.M. Smith for the notion of sets of heterogeneous relations.

APPENDIX: CROSS TERM CONDITION

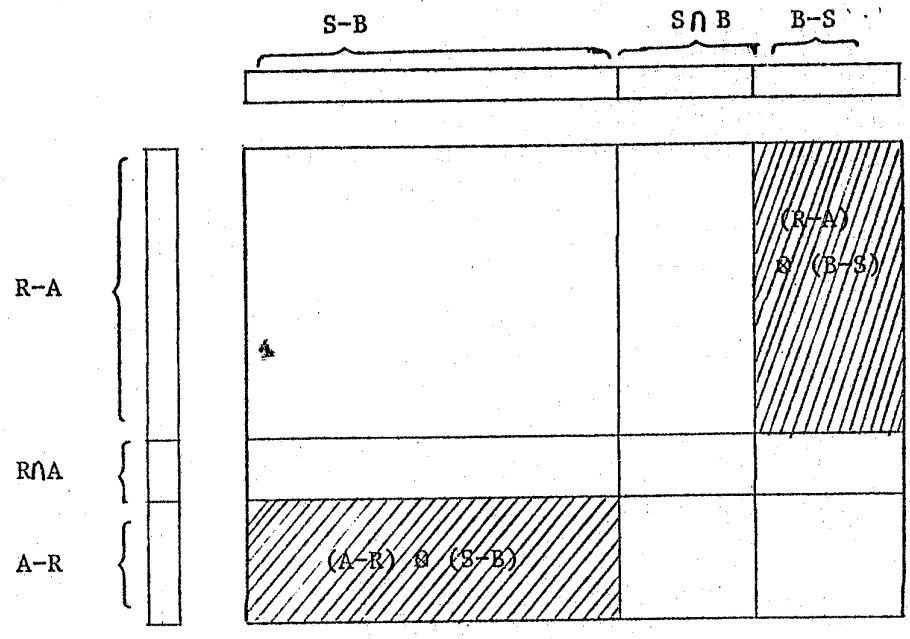
Let R be a relation with attribute set X
and S be a relation with attribute set Y
and T be a relation with attribute set $Z = X \cup Y$.

We wish to determine the conditions under which updates involving the tuples in T on the view $R \circledast S$ can be achieved by appropriate actions with tuples $A = T[X]$ and relation R, and tuples $B = T[Y]$ and relation S.

INSERTION:

If a tuple with value x for attributes X and value y for attributes Y is to appear in $R \circledast S$ after insertion, then x must be in R and y must be in S after the insertion (whether or not they were there before). Thus, in order to insert T into $R \circledast S$, it may be necessary to insert A into R and B into S.

The diagram below indicates the effect of these insertions on the product $R \circledast S$.



If T satisfies $T = A \circledast B$, then the above insertion avoids introducing spurious tuples only if $(A-R) \circledast (S-B)$ and $(R-A) \circledast (B-S)$ are simultaneously empty. The four ways in which this can happen are

$$(A-R) = (R-A) = \emptyset \Rightarrow R \equiv A$$
$$(S-B) = (B-S) = \emptyset \Rightarrow S \equiv B$$
$$(S-B) = (R-A) = \emptyset \Rightarrow (R \circledast S) \subseteq T$$
$$(A-R) = (B-S) = \emptyset \Rightarrow T \subseteq (R \circledast S)$$

Insertion of a set of tuples already contained in the product has no effect (and is harmless). Insertion of a set of tuples that contains the product is an initialization of the product. Thus whenever some tuple of $R \circledast S$ is not in T and some tuple of T is not in $R \circledast S$, the insertion is allowable if

(R=A or S=B) and $(T=A \circledast B)$.

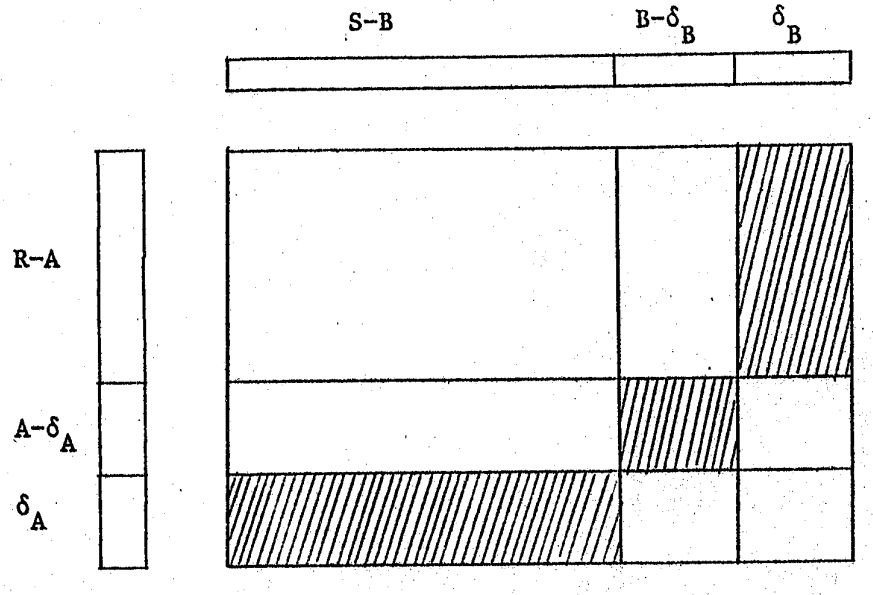
If $T \neq A \circledast B$, then the insertion is allowable only if T happens to include all the tuples in $[(A-R) \circledast S] \cup [(A-R) \circledast (B-S)] \cup [R \circledast (B-S)]$. The intricate dependence of this condition on both R and S allows us to prohibit insertions in such situations without any detrimental effect in practice. Thus, insertion of $T \not\subseteq R \circledast S$ into $R \circledast S$ is allowable if and only if

$(T = A \circledast B)$ and (R=A or S=B or $T \supseteq R \circledast S$).

DELETION:

A tuple with value x for attributes X and value y for attributes Y can be removed from $R \circledast S$ by removing either x from R or y from S.

Let δ_A be the subset of A deleted from R and δ_B the subset of B deleted from S. The diagram below indicates the effect of these deletions on the product $R \circledast S$.



Intending to delete

$$T = A \otimes B = [(A-\delta_A) \otimes (B-\delta_B)] \oplus [(A-\delta_A) \otimes \delta_B] \oplus [\delta_A \otimes (B-\delta_B)] \oplus [\delta_A \otimes \delta_B]$$

we actually delete

$$[\delta_A \otimes (S-\delta_B)] \oplus [(R-\delta_A) \otimes \delta_B] \oplus [\delta_A \times \delta_B]$$

$$= [\delta_A \otimes (S-B)] \oplus [\delta_A \otimes (B-\delta_B)] \oplus [(R-A) \otimes \delta_B] \oplus [(A-\delta_A) \otimes \delta_B] \oplus [\delta_A \otimes \delta_B].$$

Thus, we see from the diagram above that the deletion is allowable if and only if

$$(A-\delta_A) \otimes (B-\delta_B) = \delta_A \otimes (S-B) = (R-A) \otimes \delta_B = \emptyset$$

There are eight ways in which this can happen:

$$(A-\delta_A) = \delta_A = R-A = \emptyset \quad \Rightarrow \quad T = \emptyset$$

$$(A-\delta_A) = \delta_A = \delta_B = \emptyset \quad \Rightarrow \quad T = \emptyset$$

$$(A-\delta_A) = (S-B) = (R-A) = \emptyset \quad \Rightarrow \quad R \otimes S \subseteq T$$

$$(A-\delta_A) = (S-B) = \delta_B = \emptyset \quad \Rightarrow \quad \delta_A = A \;\&\; S = B \;\&\; \delta_B = \emptyset$$

$$(B-\delta_B) = \delta_A = (R-A) = \emptyset \quad \Rightarrow \quad \delta_B = B \;\&\; R = A \;\&\; \delta_A = \emptyset$$

$$(B-\delta_B) = \delta_A = \delta_B = \emptyset \quad \Rightarrow \quad T = \emptyset$$

$$(B-\delta_B) = (S-B) = (R-A) = \emptyset \quad \Rightarrow \quad R \otimes S \subseteq T$$

$$(B-\delta_B) = (S-B) = \delta_B = \emptyset \quad \Rightarrow \quad T = \emptyset.$$

Thus,if T neither contains $R \otimes S$ nor is null, then the condition for the deletion to be allowable is again

$$(T = (A \otimes B)) \text{ and } (R=A \text{ or } B=S)$$

If $T \neq (A \otimes B)$, then deletion of T is acceptable only if, for some choice of δ_A and δ_B,

$$T = (A \otimes \delta_B) \oplus (\delta_A \otimes B).$$

Disqualifying such coincidences once again, we arrive at a sufficient condition for the deletion to be allowed:

$$(T = (A \otimes B)) \text{ and } (R=A \text{ or } B=S \text{ or } T \supseteq R \otimes S),$$

which is the same as for insertions.

MODIFICATIONS:

The same argument as for deletions holds for modifications with only minor changes. Interpret δ_A as those R tuples that are changed and δ_B as the S tuples that are changed. Then intending to change tuples in $A \otimes B$, we actually change tuples in $(A \otimes \delta_B) \oplus (\delta_A \otimes B)$. Thus, the sufficient condition is identical to that for deletion. However, as well as coincidentally wanting to modify precisely the tuples $(A \otimes \delta_B) \oplus (\delta_A \times B)$, there is another way in which the modification can be valid without the sufficient condition holding:

if unintentional changes to tuples in δ_A ⊗ (S-B) and (R-A) ⊗ δ_B wind up having no effect on the set of tuples, that is for every tuple changed, some other tuple is changed to precisely what the other one used to be. Thus, in practice, the constraint is again

$$T = (A \otimes B) \text{ and } (R=A \text{ or } B=S \text{ or } T \supseteq R \otimes S)$$

Since any change to a tuple in R affects as many tuples of R ⊗ S as there are tuples in B, all the tuples in R ⊗ S with the same x values must require the same modification to those x values. Similarly, for changes to tuples of S that join with sets of tuples in R to form tuples of R ⊗ S.