FRAGMENTS OF RELATIONS

David Maier† *Oregon Graduate Center*
Jeffrey D. Ullman‡ *Stanford University*

We develop a theory of relations that are constructed by the union and selection operations from fragment relations. Algorithms for inserting and deleting from relations that are composed of physical fragments are discussed, and we show when such insertions and deletions are meaningful. We also show how to find an access set for a relation, that is, a set of fragments sufficient to produce the relation, and we apply the test to the question of how the fragmentation of relations interacts with a query on the relation, showing that a selection on the relation can be implemented by retrieving a set of physical fragments that forms an access set for another particular relation.

1. Introduction

Much has been made of the "vertical" decomposition of relations—splitting a relation into a number of components by projection, in such a way that the original relation can be recovered by joining the components. Such decompositions are at the heart of normalization and representation theory. They are used to remove redundancy and update anomalies from databases, to speed query processing, and as a means of distributing a database. Vertical decomposition enjoys a quite detailed theory.

Less attention has been accorded "horizontal" decomposition—taking subsets of the tuples in a relation through selection and recovering the relation by a union of the subsets. We shall call these subsets *fragments*, although we give the term a more restricted meaning than others. In the SDD-1 distributed database system [R*, RG], fragments are the result of both vertical and horizontal decomposition. In SDD-1, the predicates used to define the fragments are quite simple. Ceri and Pelagatti [CP], Dayal and Bernstein [DB] and Epstein, Stonebraker and Wong [ESW] have also examined fragments defined by more complex predicates than those in SDD-1.

Thus, while the theory of vertical decomposition, built from the projection and join operators, is quite general, in the sense that projections onto arbitrary sets of the original scheme are permitted, the analogous theory of horizontal decomposition, based on selection and union, has not been developed in great generality. It is the purpose of this paper to develop such a general theory of fragments.

One way we differ from previous authors concerns the disjointness condition for selection conditions; we do not assume that fragments are disjoint. Further, we do not assume that we are representing only a single relation with the fragments. Rather, we imagine that a number of different relations, albeit with the same scheme, might be defined from the fragments. We shall model a system in which there are several relations interacting by conceptually "forwarding" tuples to one another, rather than a single system controlling the allocation of tuples to fragments.

2. The Model

The scenario we envision is a set of physical relations over a single relation scheme, perhaps geographically separated, and a set of virtual relations defined by selection and union from physical relations and other virtual relations. Virtual relations may be views, or they may be relations defined as "temporaries" in a complicated expression. From this point we call all of these relations *fragments*, even though some fragments are entire relations of interest, and there might be no single relation of which all fragments are part. We illustrate such a situation with the following example. The fragments are defined over the relation scheme

$$R = \{\text{DATE, CITY, ADDR, OWNER, ARSON?}\\ \text{DAMAGES, INSURER}\}$$

The fragments contain information about structural fires in California. For this example, tuples over R will be called "fires." There are three physical fragments over R: *sf*, *la* and *arson*. The fragments *sf* and *la* reside in San Francisco and Los Angeles, respectively, and may contain fires for any city in the state, except that all San Francisco fires must be in *sf* only and all Los Angeles fires must be in *la* only. The arson fragment is maintained in Sacramento, and contains all fires t that are arsons (ARSON = "yes"). All fires from *sf* and *la* that are arsons must also be in *arson*. All arson fires that took place in San Francisco or Los Angeles must also be in *sf* or *la*, respectively. There are also three

† Work partially supported by NSF grant IST-79-18264. Part of this work was performed at SUNY, Stony Brook.
‡ Work partially supported by NSF grant MCS-80-12907.

virtual fragments:

costly: contains all fires t from *sf* or *la* where t(DAMAGES) > \$10,000

notable: contains all fires t from *costly* where t(DAMAGES) > \$100,000 and all fires t from *arson* where t(DATE) > 1 Jan 1980

recent: contains all fires t from *sf* or *la* where t(DATE) > 1 Jan 1981

We note certain behaviors that result from the definition and restrictions on these six fragments. An insertion into *sf* may require an insertion into *arson*, and an insertion to *arson* may require an insertion into *la*, but these requirements never hold at once. An insertion into *sf* never causes an insertion into *arson* that in turn causes an insertion into *la*. If we want to process a query on the fragment *recent*, and the condition t(CITY) = "San Francisco" is specified, we do not have to access the *la* fragment to construct the portion of *recent* needed to process the query. Finally, if we want to query *notable* for all fires where t(CITY) = "San Francisco" and t(ARSON?) = "yes" and t(INSURER) = "United Insurance," we have a choice of either accessing *arson* or *sf* to get the required portion of notable. The choice is important, since one or the other of these physical fragments may be at the site where the query was posed.

We now present a formalism to model situations such as the fire fragments. The model will let us deduce automatically the kinds of behaviors illustrated above, as well as give us a means to define updates on virtual fragments. We shall not distinguish physical and virtual fragments for the moment; the distinction will be made after some initial definitions. In usual relational notation, relations in a database can be distinguished by their schemes. The relation scheme can thus serve to name relations. Here, however, all relations, or fragments, have the same scheme, so we need to incorporate names for fragments into the scheme. A *fragmentation scheme* S is a quadruple $(R; \mathcal{F}, \mathcal{Q}, \mathcal{P})$ where

1. R is a set of attributes, that is, the relation scheme for all fragments,
2. $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ is the set of *fragments names*,
3. $\mathcal{Q}$ is the set of *restriction predicates*, and
4. $\mathcal{P}$ is the set of *transfer predicates*. $\mathcal{Q}$ and $\mathcal{P}$ will be explained in detail later.

A *state* f for fragmentation scheme S is a mapping from $\mathcal{F}$ to finite relations over scheme R. For $F \in \mathcal{F}$, the *F-fragment* under state f is $f(F)$. Where f is understood, we shall use "fragment F" for "F-fragment under state f." When discussing fragmentation schemes, frequently we shall use "fragment" for fragment name. The distinction should be made clear by context.

The predicates used in $\mathcal{Q}$ and $\mathcal{P}$ are necessary to restrict the set of states on the fragmentation scheme. We shall not give a specific language to express these predicates. Rather, we assume that the language is based upon the language used to express the conditions on selections in the database system at hand. We make some requirements for legal predicates:

1. Predicates have only a single free variable, which is a tuple variable over scheme R.
2. The value of a predicate on a tuple depends only on that tuple and not on the contents of any fragments. That is, the predicate can be evaluated by examining the tuple in isolation.
3. The set is closed under "and" ($\wedge$) and "or" ($\vee$).
4. Implication of predicates is decidable. That is, we can detect if $P \Rightarrow Q$ is a tautology, where P and Q are predicates.

For predicate P and tuple t, $P(t)$ denotes the value of P on t. We also use fragment names to denote predicates that represent membership in the fragment. That is, if F is a fragment name and f is a state, then $F_f(t)$, or $F(t)$ where f is understood, is true exactly when $t \in f(F)$.

We now detail the collections of predicates $\mathcal{Q}$ and $\mathcal{P}$. For each fragment name F in $\mathcal{F}$, there is *restriction predicate* Q_F in $\mathcal{Q}$.† Q_F limits the set of tuples that may appear in the F-fragment of a state. A state f *satisfies* Q_F if $F_f(t) \Rightarrow Q_F(t)$ for all tuples t. In the fires example, the *sf* fragment could have the restriction predicate t(CITY)$\neq$"Los Angeles" and the *arson* fragment could have the restriction predicate t(ARSON?)= "yes." We reiterate that the predicates in $\mathcal{Q}$ need *not* be mutually disjoint. Neither does some restriction predicate need to be true for every possible tuple over R. There may be tuples that are not permitted in any fragment. When the set F of fragment names $\mathcal{F}$ is $\{F_1, F_2, \ldots, F_n\}$, we shall abbreviate Q_{F_i} to Q_i. For each pair of fragment names F and G in $\mathcal{F}$, we may have, but are not required to have, a *transfer predicate* from F to G, P_{FG}, in $\mathcal{P}$.‡ P_{FG} indicates which tuples in the F-fragment of a state must also be in the G-fragment. State f *satisfies* P_{FG} if, for all tuples t,

$$F(t) \wedge P_{FG}(t) \Rightarrow G(t)$$

For instance, in the fires example, the transfer predicate from *sf* to *arson* could be t(ARSON?)="yes." When $\mathcal{F}$ is $\{F_1, F_2, \ldots, F_n\}$, we use P_{ij} interchangeably with $P_{F_iF_j}$.

We have a diagrammatic means to depict a fragmentation scheme $S = (R, \mathcal{F}, \mathcal{Q}, \mathcal{P})$. We give a directed graph where nodes represent fragment names and associated restriction predicates and edges represent transfer predicates. Nodes are labeled inside by a fragment name and outside by a restriction predicate:

† Strictly speaking, $\mathcal{Q}$ is a mapping that associates Q_F with F for each F in $\mathcal{F}$.

‡ Again, strictly speaking, $\mathcal{P}$ is a partial mapping from $\mathcal{F} \times \mathcal{F}$ to predicates.

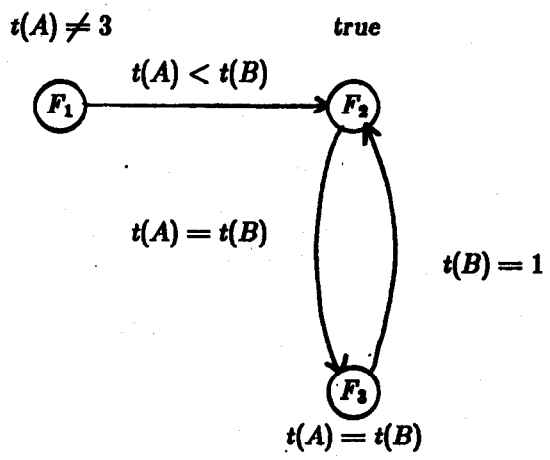
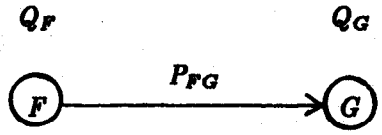
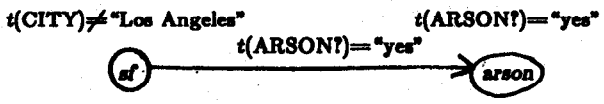


Fig. 1



A portion of the graph for the fires example is



For fragmentation scheme $S = (R, \mathcal{F}, Q, P)$, a state f for S is *legal* if it satisfies Q and P. For the fragmentation scheme in Fig. 1, where $R = \{A, B\}$, the state f where

$f(F_1) =$ A	B
1	3
2	1

$f(F_2) =$ A	B
1	3
1	1
2	2

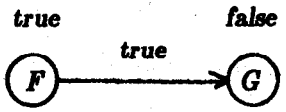
$f(F_3) =$ A	B
1	3
2	2

is legal. The state g where $g(F_1) = f(F_1)$, $g(F_2) = f(F_2)$, and $g(F_3)$ is

A	B
1	1
1	3

is not legal, both because the transfer predicate P_{23} requires tuple $<2, 2>$ to be in fragment F_3, and because Q_3 is false on $<1, 3>$.

Some fragmentation schemes only have trivial legal states. The only legal state for

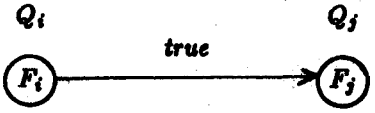


is f where $f(F) = \emptyset$ and $f(G) = \emptyset$. Notice also that there is no need for P_{ij} to be less restrictive than Q_i. Formally, if P_{ij} does not imply Q_i, then P_{ij} can be replaced by $P_{ij} \wedge Q_i$ with no effect on the set of legal states. If P_{ij} is less restrictive than Q_j, we may have an inconsistency, where a tuple is implied by P_{ij} to be in F_j, and implied by Q_j not to be in F_j. We shall avoid such inconsistencies by requiring that $P_{ij} \Rightarrow Q_j$.

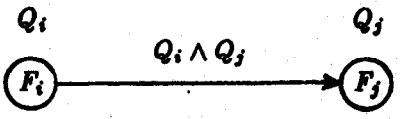
A fragmentation scheme

$$S = (R, \{F_1, F_2, \ldots, F_n\}, Q, P)$$

is *well-formed* if $P_{ij} \Rightarrow Q_i \wedge Q_j$ for $1 \leq i, j \leq n$. We assume all fragmentation schemes henceforth are well-formed. For shorthand, we allow P_{ij} to stand for $P_{ij} \wedge Q_i \wedge Q_j$. For example,



represents



It is slightly easier to see that the first diagram requires fragment F_i to be a subset of fragment F_j.

3. Physical and Virtual Fragments

We shall now distinguish between those fragments that are actually stored and those that are constructed from stored fragments. We extend a fragmentation scheme S to be quintuple $(R, \mathcal{F}, \mathcal{G}, Q, P)$ where

$$\mathcal{G} = \{G_1, G_2, \ldots, G_n\} \subseteq \mathcal{F}$$

is the set of *physical fragments*. The fragments in $\mathcal{F} - \mathcal{G}$ are *virtual fragments*. In diagrams, physical fragments will be circles and virtual fragments will be squares. The fires example is diagrammed in Fig. 2. We assume Q_{recent}, Q_{costly} and $Q_{notable}$ are the "or" of the respective incoming transfer predicates, and do not show these predicates.

This model of fragmentation seems quite powerful because of the variety of situations it can describe. For example, in addition to the complex interactions among relations illustrated in the fires example, it can be used to describe the special case where one fragment is the disjoint union of several fragments, as follows. If virtual fragment F_1 is to be the disjoint union of physical fragments F_2, F_3 and F_4, then Q_2, Q_3, and Q_4 must be mutually disjoint predicates, $Q_2 \vee Q_3 \vee Q_4 = Q_1$, $P_{1i} = Q_i$, and $P_{i1} = Q_1$, for $2 \leq i \leq 4$. A fragment F can be made a "universal" fragment—that is, F contains all the tuples from every fragment—by letting $Q_F =$*true* and $P_{GF} = Q_G$ for every other fragment G. (Actually, we shall see that the transfer predicates only need to come from the physical fragments.)

The state of a fragmentation system

$$S = (R, \mathcal{F}, \mathcal{G}, Q, P)$$

depends on values for the physical fragments. A *physical state* g for S is a mapping from $\mathcal{G}$ to relations over R. What we before called a state we shall now call a *complete state*. Complete state f *extends* physical state g if f and g agree on $\mathcal{G}$. Physical state g *corresponds* to complete state f if g is the restriction of f to $\mathcal{G}$.

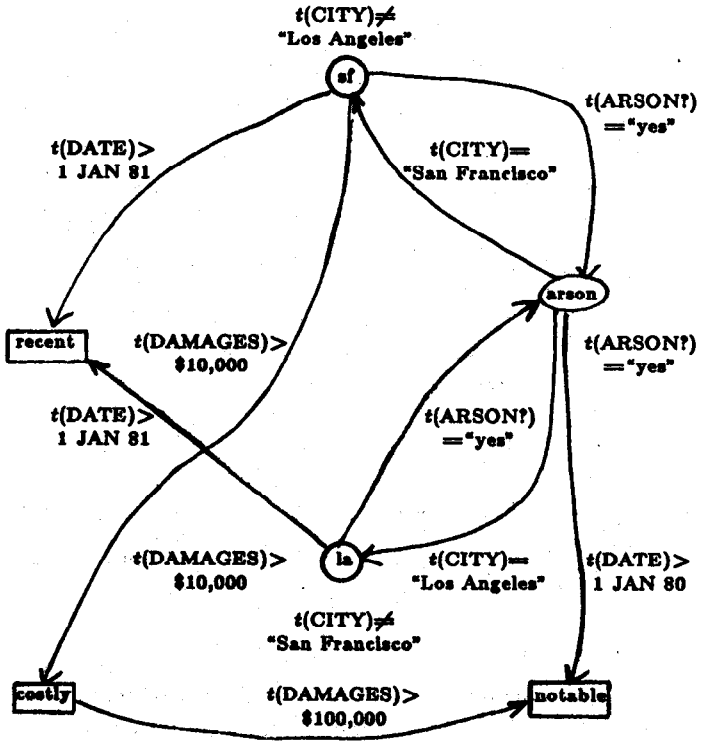


Fig. 2

Physical state g is *feasible* if there is a legal complete state f that extends g. Consider the fragmentation scheme diagrammed in Fig. 3, where $R = \{A, B\}$. In the diagram, A and B abbreviate $t(A)$ and $t(B)$. The physical state g where

$$g(F_1) = \begin{array}{|c|c|}\hline A & B \\ \hline 4 & 7 \\ 2 & 2 \\ 3 & 3 \\ \hline \end{array} \qquad g(F_2) = \begin{array}{|c|c|}\hline A & B \\ \hline 3 & 3 \\ 4 & 7 \\ \hline \end{array} \qquad g(F_4) = \begin{array}{|c|c|}\hline A & B \\ \hline 2 & 2 \\ 3 & 3 \\ 7 & 3 \\ \hline \end{array}$$

is feasible, because it can be extended to the legal complete state f where

$$f(F_3) = \begin{array}{|c|c|}\hline A & B \\ \hline 2 & 2 \\ 3 & 3 \\ \hline \end{array}$$

Note that $<7, 3>$ is not in $g(F_2)$, because the transfer predicate "$B = 3$" from F_4 to F_2 is really short for "$B = 3 \wedge (A = 3 \vee B = 4)$." However if,

$$g(F_4) = \begin{array}{|c|c|}\hline A & B \\ \hline 2 & 2 \\ 4 & 4 \\ 7 & 3 \\ \hline \end{array}$$

then g is not feasible, since any extension f would have $<3, 3>$ in $f(F_3)$ because of P_{13}, and then $<3, 3>$ would have to be in $f(F_4)$ because of P_{34}.

We want a physical state to represent a unique complete state. However, a feasible physical state may have many legal extensions. In the scheme shown in

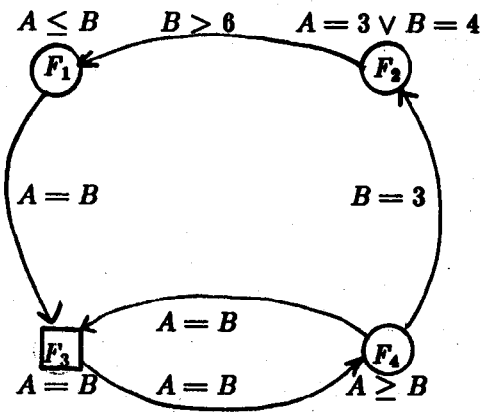


Fig. 3

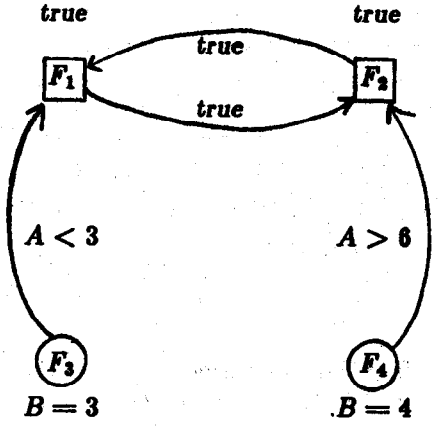


Fig. 4

Fig. 4, any physical state g satisfying Q_3 and Q_4 is feasible. Any extension of f where $g(F_3) \cup g(F_4) \subseteq f(F_1) = f(F_2)$ is legal (as are many others). We take the minimum legal extension as the complete state a physical state represents. We must show such a minimum extension always exists.

If f and f' are complete states for fragmentation scheme S, then $f \cap f'$ is the complete state for S defined by $[f \cap f'](F) = f(F) \cap f'(F)$. The following is straightforward.

Lemma 1: If f and f' are legal complete states for fragmentation scheme S that extend a physical state g for S, then $f \cap f'$ is a legal extension of g. For example, let S be the scheme in Fig. 4 and let g be the physical state where

$$g(F_3) = \begin{array}{|c|c|}\hline A & B \\ \hline 3 & 3 \\ 1 & 3 \\ \hline \end{array} \qquad \text{and} \qquad g(F_4) = \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 4 \\ 7 & 4 \\ \hline \end{array}$$

The complete states f and f' where

$f(F_1) =$ $f'(F_1) =$
$f(F_2) =$

A	B
1	3
7	4
2	2

and $f'(F_2) =$

A	B
1	3
7	4
3	3

are both legal extensions of g. Hence so is $f \cap f'$, which has

$[f \cap f'](F_1) = [f \cap f'](F_2) =$

A	B
1	3
7	4

Lemma 1 says we can define the minimum legal extension of a feasible physical state g as the legal complete state f such that for any other legal extension f', $f(F) \subseteq f'(F)$ for all fragment names F. Let $\hat{g}$ denote the minimum legal extension of a feasible physical state g. There is a (conceptually) simple algorithm to determine if a physical state g is feasible, and if so, to find $\hat{g}$. We initialize $\hat{\mathbf{g}}$ so that $\hat{\mathbf{g}}(G) = g(G)$ for $G \in \mathcal{G}$ and $\hat{g}(F) = \emptyset$ for $F \in \mathcal{F} - \mathcal{G}$. Considering all pairs of fragments F and F', we add to $\hat{g}(F')$ all tuples t such that $t \in \hat{g}(F)$ and $P_{FF'}(t)$ is true. If at any point we add a new tuple to a physical fragment G, we know g is not feasible. Otherwise, we keep considering pairs of fragments until no changes can be made, at which point $\hat{g}$ has the correct value.

Theorem 1: The above algorithm correctly computes $\hat{g}$.

Proof: An easy induction on the number of tuples added to relations shows that each must be present in any extension of g. Since nothing else is added to any relation, the result must be minimal. If we try to add a tuple to a physical relation, then it too must be there, since if it is not, there can be no legal extension of g.

4. The Complete Transfer Matrix

We can compute $\hat{g}$ given a feasible physical state g. It does not seem likely that circumstance would demand that all of $\hat{g}$ be computed at once. It seems more likely that a particular query on a fragmented database would only be interested in $\hat{g}(F)$ for a single fragment F. We would like to optimize the calculation of $\hat{g}(F)$. Especially, we would like to know when we could get by with consulting less than all of the physical fragments to determine the value of $\hat{g}(F)$. Given fragmentation scheme $S = (R, \mathcal{F}, \mathcal{G}, \mathcal{Q}, \mathcal{P})$ and fragment name F in $\mathcal{F}$, an *access set* A for F is a subset of $\mathcal{G}$ such that $\hat{g}(F)$ can always be found using a fixed computation on the fragments $g(G)$ for G in A.

Obviously, $\{G\}$ is always an access for any physical fragment G, although access sets for G not including G could exist. We are more interested in access sets for virtual fragments. We shall not attempt to characterize a "best" access set for a fragment, since our model does not deal with the actual configuration of hardware supporting the fragmentation scheme. Surely a "best" access set depends on the allocation of physical fragments to processors, the point at which the fragment is being constructed, the communication costs between processors, and how fast predicates can be evaluated on each processor. Our model does not incorporate any of these factors. We simply assume there is some way of evaluating the desirability of different access sets, and concentrate on determining when a set $A \subseteq \mathcal{G}$ is an access set for a given fragment.

To help us with this problem we define what is essentially a transitive closure of a set of transfer predicates. Let $S = (R, \mathcal{F}, \mathcal{G}, \mathcal{Q}, \mathcal{P})$ be a fragmentation scheme where $F = \{F_1, F_2, \ldots, F_n\}$. The *transfer matrix* C for S is an $n \times n$ matrix of predicates. The $i-j$ entry of C, C_{ij}, is the weakest predicate such that $F_i(t) \wedge C_{ij}(t) \Rightarrow F_j(t)$ for all legal complete states of S. The transfer matrix tells us directly which tuples in a fragment F must also be in a fragment G in a legal state. Transfer matrices will also let us define insertions and deletions on virtual fragments and help to optimize queries. For example, in Fig. 3, C_{14} is $A = B$, C_{12} is $A = B = 3$ and C_{41} is *false*. We give a constructive definition of C. We shall assume the transfer predicate P_{ij} exists for all $1 \leq i, j \leq n$. Let $P_{ii} = Q_i$ and let P_{ij}=*false* if it is not explicitly mentioned in $\mathcal{P}$. Let a *path* from fragment F_{i1} to fragment F_{ik} be any sequence of fragment names $F_{i1}, F_{i2}, \ldots, F_{ik}$. The *label* of such a path p is given as

$$L_p = P_{i_1 i_2} \wedge P_{i_2 i_3} \wedge \cdots \wedge P_{i_{k-1} i_k}$$

The label of a path from F to G is essentially the transfer predicate from F to G along that path. The *cost* from F_i to F_j is the disjunction of all the labels of paths from F_i to F_j,

$$\bigvee_{\substack{p \text{ a path} \\ \text{from } F_i \text{ to } F_j}} L_p$$

It is the cost from F_i to F_j that is the C_{ij} entry of the transfer matrix for S.

Theorem 2: C_{ij} computed as above is the weakest predicate C for which $F_i(t) \wedge C(t)$ implies $F_j(t)$.

Proof: It is easy to check that each L_p, and therefore the disjunction of the L_p's, satisfies the desired condition, $F_i(t) \wedge C_{ij}(t) \Rightarrow F_j(t)$.

Conversely, suppose there is a C satisfying $F_i \wedge C \Rightarrow F_j$, yet there is a tuple t for which $C(t)$ is true, while $C_{ij}(t)$ is false. Start with a state h in which F_i contains t, and all other virtual and physical fragments are empty. Extend h by inserting tuples into whatever fragments we must, using the same algorithm that we use to extend physical states to complete states. Since $C(t)$ and $F_i(t)$ are both true, $F_j(t)$ must be true in the extension of state h. Following the extension algorithm, we see that there must be some path from F_i to F_j that

forces the presence of t in F_j. For this path p, $L_p(t)$ holds, so $C_{ij}(t)$ holds. The set of permissible predicates is a closed semiring in the sense of [AHU], where $\vee$ is addition, $\wedge$ is multiplication, *false* is the additive identity and *true* is the multiplicative identity. There is in [AHU] a general algorithm for computing pairwise costs in a digraph, where the edges are labeled by elements from a closed semiring. Suitable choices for the semiring yield algorithms for computing a regular expression from a finite automation, the transitive closure of a digraph, and shortest paths in a network. Using our semiring of predicates, we get an algorithm for computing the entries in a transfer matrix. This specialization of the algorithm follows; it uses auxiliary predicate-valued variables $\{C_{ij}^k \mid 1 \le i,j \le n, 0 \le k \le n\}$.

for $i := 1$ **to** n **do** $C_{ii}^0 := P_{ii}$;
for $k := 1$ **to** n **do**
  for all $1 \le i,j \le n$ **do**
    $C_{ij}^k := C_{ij}^{k-1} \vee (C_{ik}^{k-1} \wedge (C_{kk}^{k-1})^* \wedge C_{kj}^{k-1})$
for all $1 \le i,j \le n$ **do**
  $C_{ij} := C_{ij}^n$;

The * here is the closure operator, defined as

$$P^* = \mathit{false} \vee P \vee (P \wedge P) \vee (P \wedge P \wedge P) \vee \cdots$$

Clearly $P^* = P$ in this semiring. It turns out that C_{kk}^ℓ is always P_{kk}, which is Q_k, so that innermost assignment statement can be replaced by

$$C_{ij}^k := C_{ij}^{k-1} \vee C_{ik}^{k-1} \wedge Q_k \wedge C_{kj}^{k-1}$$

Intuitively, this simplification means that we need not consider transfer predicates along paths that cross themselves. The label for such a path is always subsumed by the label of the corresponding path without the cycle. We note a few properties of the transfer matrix and its computation.

Theorem 3: If the fragmentation system is well-formed, i.e., $P_{ij} \Rightarrow Q_i \wedge Q_j$ for all i and j, then the transfer matrix will be well-formed. That is, $C_{ij} \Rightarrow Q_i \wedge Q_j$ for all i and j.

Proof: This observation is easily proved inductively with the hypothesis $C_{ij}^k \Rightarrow Q_i \wedge Q_j$. The transfer matrix can be used to check feasibility of a physical state g. Testing $F_i(t) \wedge C_{ij}(t) \Rightarrow F_j(t)$ for all pairs of physical fragments F_i and F_j suffices.

Theorem 4: The value of a virtual fragment F_j under complete state $\hat{g}$ can be expressed with the transfer matrix by

$$F_j(t) = \bigvee_{F_i \in \mathcal{G}} F_i(t) \wedge C_{ij}(t)$$

Proof: Theorem 2 tells us we do not compute too much on the right side, since G_{ij} is the weakest possible transfer predicate. Theorem 1 tells us the right side is not too little, since tuples belong in F_j if and only if some path transfers them there from a physical fragment.

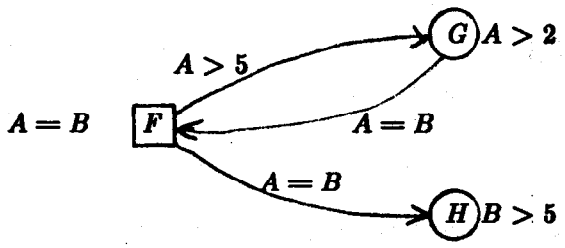


Fig. 5

The above equation can be interpreted in relational algebra as

$$\hat{g}(F_j) = \bigcup_{F_i \in \mathcal{G}} \sigma_{C_{ij}}(g(F_i))$$

5. Insertion and Deletion

The principle we espouse for defining insertions and deletions to fragments is that a request to update a fragment $\hat{g}(F)$ implies whatever changes are necessary to g to transform it to a feasible physical state h, where $\hat{h}(F)$ reflects the desired change.

Insertion of a tuple t into a physical fragment G is fairly simple to describe. Tuple t is added to G and to any other physical fragment H such that $C_{GH}(t)$=*true*. (Here we are using $C_{F_iF_j}$ for C_{ij}.) Insertion into a virtual fragment cannot be defined in the same manner, since there is no guarantee that the change in the virtual fragment will cause any change in the corresponding physical state. For example, in Fig. 5, inserting the tuple <4, 4> into virtual fragment F necessitates no change in the physical state, hence the tuple will not be present when F is next constructed from the physical fragments.

Another problem is that a tuple t inserted into some virtual fragment F may also be transferred to physical fragment G because $C_{FG}(t)$=*true*, but may have $C_{GF}(t)$=*false*, so the tuple is not transferred back when F is constructed from the physical state. To insert tuple t into a virtual fragment F, since we want such an insertion to have an effect, we must place the tuple in some physical fragment G such that $C_{GF}(t)$=*true*. Thus insertion of tuple t into fragment F has two steps, possibly.

1. First, t is added to every physical fragment G where C_{FG}=*true*. Say these insertions cause the physical state to change from g to h.
2. If $\hat{h}(F)$ does not contain t, then t is inserted into some physical fragment G where $C_{GF}(t)$ =*true* The insertion into G also entails updates in physical fragments to which G must transfer t.

This two-step procedure has the desired effect on F while preserving legality of complete states. However, there may be no such G into which to insert t. We

define a virtual fragment F as *insertable* if

$$Q_F(t) \Rightarrow \bigvee_{G \in \mathcal{G}} C_{GF}(t)$$

That is, in any complete state, any tuple that can legally belong to fragment F can be placed in some physical fragment that will transfer the tuple to F. All physical fragments G are by definition insertable, since $C_{GG} = P_{GG} = Q_G$. Note that a virtual fragment F can always be made insertable by strengthening Q_F.

Although the nondeterministic nature of the insertion algorithm at step (2) may seem undesirable, it does give leeway for optimizing insertion requests. Going back to the fires example, an insertion into *recent* for a fire in San Jose can be carried out on *sf* or *la*, depending on the site of the update request. However, it is possible to test if there will ever be any choice in updating a given virtual fragment. We define a virtual fragment F to be *deterministic* if

$$Q_F(t) \Rightarrow \bigvee_{G \in \mathcal{G}} C_{FG}(t) \wedge C_{GF}(t)$$

That is, any tuple t that can belong to F must be transferred to some physical fragment G that transfers t back to F.

Deletion can be defined uniformly for all fragments. A request to delete tuple t from fragment F causes t to be removed from any physical fragment G where $C_{GF}(t)$=*true*. Such a transformation has the required effect on F while preserving legality of the complete state. However, we have no guarantee that inserting tuple t into fragment F and deleting tuple t from fragment F are necessarily inverses as far as the physical state of the database is concerned. We define a fragment F to be *faithful* if for every physical fragment G, $C_{FG}(t) \Rightarrow C_{GF}(t)$. That is, if insertion of tuple t into F necessitates insertion of t into G, then deletion of t from F will result in the deletion of t from G.

We can summarize the above points in the following three theorems, each of which is a straightforward application of definitions.

Theorem 5: Insertion of an arbitrary tuple t satisfying Q_F into virtual fragment F can be effected by insertion into one or more physical fragments if and only if F is insertable.

Theorem 6: Let F be an insertable fragment. Then our proposed insertion algorithm never causes us to choose an extra physical fragment into which t is inserted at step (2) if and only if F is deterministic.

Theorem 7: If fragment F is faithful, then insertions and deletions from F are inverse operations on the state.

Note that our algorithm for constructing the C_{FG}'s gives us an efficient test for each of these properties, modulo the efficiency of tautology testing. That is, if we assume that for the sorts of transfer functions encountered in practice, recognizing tautologies will not be too difficult, then the entire algorithm will run in an acceptable length of time. In the most general cases, we cannot even decide tautologies, so we cannot assert a theorem guaranteeing performance of the algorithm.

6. Querying Fragments: Construction and Selection

An important use for the concepts we have developed is in optimization of queries about virtual fragments. Our transfer predicates generalize disjoint unions, and in effect allow us to create a generalized union operator for an extended relational algebra. In particular, selections on virtual fragments can be integrated with the notation already developed, allowing us to implement such selections by a minimal set of retrievals on physical relations.

In the simplest case of querying a virtual fragment, where we want the entire fragment, we need to construct the fragment from the physical state. As we have noted, an access set for a virtual fragment can be smaller than the set of all physical fragments. We examine here how to test whether a set A of physical fragments is actually an access set for a virtual fragment F. In the more complicated case where a query involves a selection from a virtual fragment, we could construct the entire fragment and then do the selection. However, prior examination of the selection predicate may allow us to use a partial construction (one that does not access all physical fragments in an access set) of a virtual fragment to be used to evaluate the selection query. We show that determining which physical fragments will suffice for a given selection reduces to the problem of testing for an access set.

Given a feasible physical state g, we have seen that for a virtual fragment F, $\hat{g}(F)$ is formed by taking all the tuples in each physical fragment G that satisfy the complete transfer predicate C_{GF}. However, if we know for some physical fragment $H \in \mathcal{G}$ that any tuple that H transfers to F is also present in some other physical fragment that will transfer the tuple to F, then H need not be accessed to construct $\hat{g}(F)$. More formally, we have the following.

Lemma 4: Let $S = (R, \mathcal{F}, \mathcal{G}, \mathcal{Q}, P)$ be a fragmentation scheme where $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$, and let $C = [C_{ij}]$ be the complete transfer matrix for S. Let F be a virtual fragment with an access set $A = \{G, H_1, H_2, \ldots, H_k\} \subseteq \mathcal{G}$. $A' = A - \{G\}$ is an access set for F if and only if

$$C_{GF}(t) \Rightarrow \bigvee_{H \in A'} C_{GH}(t) \wedge C_{HF}(t)$$

Proof: If the above is not satisfied for some tuple t, then we can place t in G and F, yet not have t appear in

any of $H_1, \ldots, H_k$ that forces t to be in F. Therefore, we shall not be able to deduce from A' the presence of t in F.

To test whether a set $A \subseteq \mathcal{G}$ is an access set for a virtual fragment F, start with $\mathcal{G}$ and eliminate fragments in $\mathcal{G} - A$ one at a time. If each fragment removed satisfies the condition of Lemma 4 with respect to the remaining fragments, then A is an access set for A.

We are now prepared to apply the above observation to the problem of optimizing queries involving selection on a virtual fragment. We can view the result of a selection on a virtual fragment as another virtual fragment. If we want $\sigma_S(F)$ for virtual fragment F, where S is the selection predicate, we can imagine a new virtual fragment F' in the fragmentation scheme. We let $Q_{F'} = true$ and add the transfer predicate $P_{FF'} = S \wedge Q_F$, as shown below.

rest of scheme	$S \wedge Q_F$	$true$
F		F'

Determining what physical fragments to access to compute $\sigma_S(F)$ is now the problem of determining an access set for F' in this augmented scheme. Let $C = [C_{ij}]$ be the complete transfer matrix for the original fragmentation scheme. We need the complete transfer matrix C' for the augmented scheme. Since any transfer from the rest of the scheme to F' must be through F, C' can be computed directly from C by

1. $C'_{GH} = C_{GH}$ for $G, H \neq F'$,
2. $C'_{F'F'} = true$;
3. $C'_{F'G} = false$ for $G \neq F'$;
4. $C'_{GF'} = C_{GF} \wedge S \wedge Q_F$ for $G \neq F'$.

Having found an access set A for F', it is straightforward to compute

$$\sigma_S(F) = \bigcup_{G \in A} \sigma_S(G)$$

7. Further Questions

Our model cannot express qualifications such as "Every tuple in fragment F must be in fragment G_1 or in fragment G_2." Another qualification that cannot be expressed is "Fragments F_1 and F_2 are always disjoint," unless that condition happens to follow from the restriction predicates for F_1 and F_2. Are there simple extensions of the model that will allow such qualifications and still yield simple algorithms for insertion and deletion into virtual fragments, commutativity of "union" with selection, and the other useful properties?

Bibliography

[AHU] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[CP] S. Ceri and G. Pelagatti. "Correctness of execution strategies of read-only transactions in distributed databases." Instituto di Elettrotecnica ed Elettronica Politecnico di Milano internal report 80-16.

[DB] U. Dayal and P. A. Bernstein, "The fragmentation problem: Lossless decomposition of relations into files." Computer Corporation of America TR CCA-78-13, November 1978.

[ESW] R. Epstein, M. Stonebraker and E. Wong. "Distributed query processing in a relational database system." *Proc. 1978 ACM-SIGMOD International Conf. on Management of Data*, May-June 1978, pages 169-180.

[P] J. Paredaens. "Horizontal and vertical decompositions." Presented at XP1 Workshop on Relational Database Theory, Stony Brook, N.Y., June-July 1980.

[R*] J. B. Rothnie, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. "Introduction to a system for distributed databases (SDD-1)." *ACM TODS* 5:1, 1980, pages 1-17.

[RG] J. B. Rothnie and N. Goodman. "An overview of the preliminary design of SDD-1: A system for distributed databases." *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977, pages 39-57.