HORIZONTAL DECOMPOSITIONS AND THEIR IMPACT ON QUERY SOLVING

by P. De Bra and J. Paredaens,
Dept. of Mathematics, University of Antwerp, Universiteitsplein 1, B-2610 Wilrijk.

1. INTRODUCTION

The vertical decomposition of relations into projections of these relations improves the efficiency and the security of data manipulations. [Fa,Ul] However, not all manipulations or answers to queries can be improved by this decomposition, and it is the given set of constraints (associated with the relation scheme) that determines of which actions the efficiency is improved.

In this paper we show how to bypass the correspondence between the dependency structure and the possible vertical decompositions of a relation scheme, by performing a horizontal decomposition first. [De,Pa]

For some queries it is easy to find out which functional dependencies (fd's) have to be satisfied to allow a vertical decomposition which improves the efficiency of answering that query (if necessary). Such fd's can be represented by "goals", which are couples of sets of attributes, e.g. (X,Y). The goals, associated with a scheme, represent the "desired" dependency structure, which may differ from the actual dependency structure. By decomposing the scheme horizontally a number of subschemes are obtained which (partially) satisfy the desired fd's.

2. AFUNCTIONAL DEPENDENCIES

The horizontal decomposition is based on the following definition :
A set of tuples S, in an instance, is called X-complete (for a set X of attributes) iff the tuples not belonging to S all have other X-values than those belonging to S. Formally : for all $t_1 \in S$, $t_2 \notin S$ holds $t_1[X] \neq t_2[X]$.

The horizontal decomposition of an instance R, according to a goal (X,Y), is the couple of instances (R_1,R_2), where R_1 is the greatest X-complete subset of R in which the fd $X \rightarrow Y$ holds, and where $R_2 = R-R_1$.

In R_2 the so-called afunctional dependency (ad) $X \not\rightarrow Y$ holds, which means that in every X-complete subset of R_2 the fd $X \rightarrow Y$ does not hold.

From now on we let a scheme R have a set A of ad's as well as a set $\mathcal{F}$ of fd's. The horizontal decomposition of a scheme R, according to the goal (X,Y) is the couple of schemes (R_1,R_2) where R_1 has fd's $\mathcal{F} \cup \{X \rightarrow Y\}$ and R_2 has fd's $\mathcal{F}$ and the ad $\{X \not\rightarrow Y\}$. For every instance R of R, R_1 is the largest

X-complete subset of tuples in which $X \rightarrow Y$ holds and R_2 is the largest X-complete subset in which $X \not\rightarrow Y$ holds.

To illustrate this horizontal decomposition, consider the following example :

Let PARKING (E,PB) be a (part of a) database scheme, representing Employees, hiring Parking Boxes for their car(s). A reasonable query for this database is : "How many parking boxes are hired by employee e ?" If every employee would not be allowed to hire more than one box, this query could be solved faster than when this constraint is not satisfied. However, although most employees hire only one box, some (upper class) employees may want to hire more than one box. Hence the "desired" fd $E \rightarrow PB$ does not hold in the scheme PARKING.

By decomposing PARKING according to (E,PB) into PARKING1, representing the employees who hire only one box (and their box), and PARKING2, representing the employees who hire several boxes, (and their boxes), part of the efficiency that was lost, because of the violation(s) of $E \rightarrow PB$, can be regained.

3. A MEMBERSHIP ALGORITHM

The horizontal decomposition of a scheme, according to (X,Y), is only useful if neither $X \rightarrow Y$ nor $X \not\rightarrow Y$ must hold in the relation scheme. Hence a membership algorithm is needed for (mixed) fd's and ad's. Since for fd's the membership problem is well known we only present a membership algorithm for ad's (but considering the presence of fd's too).

We first introduce the notion of "conflict". A set $\mathcal{F} \cup A$ of fd's and ad's is said to be in conflict iff only the empty set of tuples satisfies all dependencies of $\mathcal{F} \cup A$. It should always be kept in mind that the set of dependencies, associated with a database scheme, must never be in conflict. To verify whether $\mathcal{F} \cup A$ is in conflict one may use the following algorithm :

Algorithm 3.1. : conflict detection
Input : a set $\mathcal{F}$ of fd's and a set A of ad's.
Output : $\mathcal{F} \cup A$ is in conflict or not.
Method :
if for some $T \not\rightarrow U \in A$ holds $T \rightarrow U \in \mathcal{F}^*$ {$\mathcal{F}^*$ contains all the consequences of $\mathcal{F}$}
then $\mathcal{F} \cup A$ is in conflict
else $\mathcal{F} \cup A$ is not in conflict. □

In [De,Pa] this algorithm, and also the following one, have been proved.

Algorithm 3.2. : membership of ad's.

Input : a set $\mathcal{F}$ of fd's, a set A of ad's, and an ad $X \twoheadrightarrow Y$.
$\mathcal{F} \cup A$ is assumed not to be in conflict.

Output : $X \twoheadrightarrow Y \in (\mathcal{F} \cup A)^*$ or $X \twoheadrightarrow Y \notin (\mathcal{F} \cup A)^*$ ($(\mathcal{F} \cup A)^*$ contains all the consequences of $\mathcal{F} \cup A$).

Method :

if $\mathcal{F} \cup \{X \rightarrow Y\} \cup A$ is in conflict {use algorithm 3.1.}
then $X \twoheadrightarrow Y \in (\mathcal{F} \cup A)^*$
else $X \twoheadrightarrow Y \notin (\mathcal{F} \cup A)^*$ □

From the above algorithms it is obvious that the membership algorithm has time complexity $O(m.f(n))$, where $n = \# \mathcal{F}$, $m = \# A$ and f is the function that indicates the time-complexity of the membership algorithm for fd's, used in algorithm 3.1.

4. THE INHERITANCE PROBLEM.

In section 3 it was indicated that before decomposing a scheme R according to (X,Y) one must verify whether neither $X \rightarrow Y$ nor $X \twoheadrightarrow Y$ holds in R. Hence, if one wants to decompose the subschemes further on (according to other goals), it is necessary to determine which dependencies hold in the subschemes. This is called the inheritance problem.

In the PARKING example the inheritance problem does not occur since (E,PB) is the only goal.

For the traditional vertical decomposition into Boyce-Codd normal form, the inheritance problem is easy to solve. However, in [Be] it is proved that the performance of one vertical decomposition step probably must take exponential time. (A BCNF-test is NP-complete)

In [De] the inheritance problem for the horizontal decomposition (according to goals) is proved to be better behaved :
In R_1 $(\mathcal{F} \cup (X \rightarrow Y\} \cup \hat{A})^*$ holds and in R_2 $(\mathcal{F} \cup \hat{A} \cup \{X \twoheadrightarrow Y\})^*$ holds, where $\hat{A} = \{T \twoheadrightarrow U \in A \mid T \rightarrow X \in \mathcal{F}^*\}$. Hence, to obtain (a generating set for) the sets of fd's and ad's that hold in the subschemes, it is not necessary to calculate the closure of any set of dependencies. The performance of one decomposition step takes $O(\ell.m.f(n))$ time, where $\ell = \# G$, $m = \# A$, $n = \# \mathcal{F}$ and f determines the time complexity of the fd-membership algorithm.

5. A DECOMPOSITION ALGORITHM

The decomposition according to goals involves the danger of infinite decompositions. If the number of times that a goal may be used is not bound , it is very easy to generate infinite decompositions. Even in the PARKING example, adding the goal (PB,E) to {(E,PB)} would cause an infinite decomposition. To avoid this problem we simply restrict the use of goals by avoiding them to be used repeatedly. This is established in the next algorithm.

Algorithm 5.1. : decomposition algorithm.

Input : A scheme R with fd's $\mathcal{F}$, ad's A and goals G.

Output : If $\mathcal{F} \cup A$ is not in conflict, a decomposition $(R_1, R_2, \ldots)$ of R.

Method :

```
procedure decompose (R;F : set of fd's, A : set of ad's, G : set of goals);
begin
if for all (X,Y) ∈ G holds X → Y ∈ F* or X → Y ∈ (F ∪ A)*
   then the decomposition of R is (R)
   else  begin {let (X,Y) be a goal for which neither X → Y nor X ↛ Y holds}
         Â: = φ;
         for each T ↛ U ∈ A
         do if T → X ∈ F*
               then Â: = Â ∪ {T ↛ U};
         decompose (R1, F ∪ {X → Y}, Â, G-{X,Y});
         decompose (R2, F, Â ∪ {X ↛ Y}, G-{X,Y});
         the decomposition of R is (the decomposition of R1, the decomposition
         of R2)
         end;
end;
begin
if F ∪ A is not in conflict
   then decompose (R,F,A,G)
   else produce an error message
end.                                                                    □
```

All subschemes (except one), produced by algorithm 5.1., partially satisfy the fd's $X \rightarrow Y$ for which $(X,Y) \in G$. One subscheme satisfies all these fd's, and one other subscheme satisfies none of them.

The decomposition, defined by algorithm 5.1. is "fd-preserving" (but not "ad-preserving"). Hence the traditional vertical decomposition can be performed on the (final) subschemes, resulting from algorithm 5.1.

6. CONCLUSIONS

A new way to decompose relations has been given. An algorithm is proposed, which can be used as a pre-processor for any algorithm for vertical decomposition according to fd's (e.g. decomposition into third normal form or Boyce-Codd normal form). Furthermore the horizontal decomposition can be implemented much more efficiently than the vertical decomposition.

In future investigations an attempt should be made to extend the horizontal decomposition to cover multivalued and join-dependencies as well as functional dependencies.

REFERENCES.

[Be] Beeri C., Bernstein P.A., Computational Problems Related to the Design of Normal Form Relation Schemes, ACM TODS 1979, 4:1, pp 30-59.

[De] De Bra P., Paredaens J., The Membership and the Inheritance of Functional and Afunctional Dependencies, Dept. of Mathematics, Univ. of Antwerp, Belgium, report 81-39.

[Fa] Fagin R., Normal forms and relational database operators, Proceedings ACM SIGMOD (1979), pp. 153-160.

[Pa] Paredaens J., De Bra P., On Horizontal Decompositions, XP2 Congress, State University of Pennsylvania, June 1981.

[Ul] Ullman J., Principles of Data Base Systems, Pitman, 1980.