Information Processing Letters 17 (1983) 91–95
North-Holland

24 August 1983

AN ALGORITHM FOR HORIZONTAL DECOMPOSITIONS

Paul De BRA and Jan PAREDAENS

Department of Mathematics, University of Antwerp, B-2610 Wilrijk, Belgium

Communicated by W.L. Van der Poel
Received June 1982

Keywords: Relational Database Model, functional dependencies, horizontal decompositions, inherited dependencies, normal forms

Introduction

In the study of the Relational Database Model, the vertical decomposition of relations into projections of these relations has been emphasized ever since its introduction in [3]. The presence (in the 'real world') of a constraint, e.g., a functional dependency, is indispensable for the use of vertical decompositions.

However, in many applications it is necessary to allow violation of the desired constraints. In this paper an 'exception handling mechanism' is described which is based on the horizontal decomposition of a relation into restrictions of this relation, using the union as composition operator. By separating the 'exceptions' from the main part of the relation, the desired functional dependency is preserved in this main part. Hence, vertical decomposition of (part of) the relation becomes possible after performing the horizontal decomposition.

To illustrate the use of horizontal decompositions consider a (small part of a) database PARKING (E, PB, C), representing *E*mployees hiring *P*arking *B*oxes for their *C*ar(s). Most employees only have one car and only hire one parking box. This, however, is not a constraint that might be used for any optimization of the database. The horizontal decomposition, described below, selects a (large) subrelation in which every employee only has one car and only hires one box. The latter constraint (a functional dependency) is very useful for optimizing the database structure.

1. The relational model

In Codd's relational database model [3,7], a relation *instance* is a table in which each column corresponds to a distinct *attribute*, and each row (i.e., *tuple*) to a distinct entity. For each attribute there is a set of possible values, called the *domain* of that attribute.

The relation instance (the set of tuples) changes with time, whereas its structure is static. This structure, the name of the relation, its attributes, its domains and the constraints that must be satisfied in every instance (i.e., at any time), is called the (relation) *scheme*.

Let X and Y be sets of attributes. The *functional dependency* (fd) $X \rightarrow Y$ means that (in every instance) iff two tuples s and t have the same value on X ($s[X] = t[X]$), then they also have the same value on Y ($s[Y] = t[Y]$).

2. Horizontal decompositions

A set of tuples S, in an instance, is called X-*complete* iff the tuples not belonging to S all

have other X-values than those belonging to S.

Let X, Y be sets of attributes. The *horizontal decomposition* of an instance R, according to the *goal* (X, Y) is the couple of (sub)instances (R_1, R_2), where R_1 is the largest X-complete set of tuples (of R) in which the fd $X \rightarrow Y$ holds, and $R_2 = R - R_1$ (the exceptions to $X \rightarrow Y$).

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

Let $\mathcal{R}(\mathcal{F}, \mathcal{A})$ be a relation scheme, with set of fd's $\mathcal{F}$ and set of ad's $\mathcal{A}$. The horizontal decomposition of $\mathcal{R}$ according to (X, Y) is

$$\left(\mathcal{R}_1\left(\mathcal{F} \cup \{X \rightarrow Y\}, \hat{\mathcal{A}}\right), \mathcal{R}_2\left(\mathcal{F}, \hat{\mathcal{A}} \cup \{X \not\rightarrow Y\}\right)\right),$$

where $\hat{\mathcal{A}}$ is a subset of $\mathcal{A}$, of which the calculation is described in Section 4.

Recall the PARKING example of the Introduction. The presence of the 'almost functional dependencies' suggests the use of the goals (E, PB) and (E, C) for horizontal decomposition. (E, PB) results in

$$\text{PARKING}_1(\{E \rightarrow PB\}, \emptyset),$$

and

$$\text{PARKING}_2(\emptyset, \{E \not\rightarrow PB\}).$$

Decomposing the subschemes further on, using (E, C) gives

$$\text{PARKING}_{11}(\{E \rightarrow PB, E \rightarrow C\}, \emptyset),$$

$$\text{PARKING}_{12}(\{E \rightarrow PB\}, \{E \not\rightarrow C\}),$$

$$\text{PARKING}_{21}(\{E \rightarrow C\}, \{E \not\rightarrow PB\})$$

and

$$\text{PARKING}_{22}(\emptyset, \{E \not\rightarrow PB, E \not\rightarrow C\}).$$

Consider the following instance of PARKING:

PARKING =

E	PB	C
Jones	1	FIAT 500
Smith	2	RABBIT
Harvey	3	ESCORT
Harvey	3	GRANADA
Johnson	4	SILVERGHOST
Johnson	5	MERCEDES 600

After the decomposition we have the instances:

PARKING_{11} =

E	PB	C
Jones	1	FIAT 500
Smith	2	RABBIT

PARKING_{12} =

E	PB	C
Harvey	3	ESCORT
Harvey	3	GRANADA

$\text{PARKING}_{21} = \emptyset$

PARKING_{22} =

E	PB	C
Johnson	4	SILVERGHOST
Johnson	5	MERCEDES 600

For obvious reasons we have left PARKING_{21} empty. To express that this relation should be empty, i.e., that an employee who only owns one car may only hire one parking box, a new constraint would be necessary [5], which is not discussed in this paper.

3. A membership algorithm

For fd's a number of membership algorithms can be found in the literature [1,2]. It is very easy to prove that the presence of ad's does not affect the membership of fd's.

Some sets of fd's and ad's $\mathcal{F} \cup \mathcal{A}$ cannot be satisfied in a nonempty instance, for instance if

$$\mathcal{F} = \{A \rightarrow B, B \rightarrow C\} \quad \text{and} \quad \mathcal{A} = \{A \not\rightarrow C\}.$$

They are said to be *in conflict*. The set of dependencies, associated with a database scheme must never be in conflict. To verify whether $\mathcal{F} \cup \mathcal{A}$ is in conflict one may use the following algorithm, which relies on [4, Theorem 4.1].

Algorithm 3.1. Conflict detection

Input : A set $\mathcal{F}$ of fd's and a set $\mathcal{A}$ of ad's.
Output : $\mathcal{F} \cup \mathcal{A}$ is in conflict or not.
Method:
begin
for each $T \not\rightarrow U \in \mathcal{A}$ **do**

if $\mathcal{F} \models T \to U$ {i.e., if $T \to U$ is a consequence of $\mathcal{F}$. This can be verified using an algorithm of [1,2]}.
then return "$\mathcal{F} \cup \mathcal{A}$ is in conflict"; {and stop}
return "$\mathcal{F} \cup \mathcal{A}$ is not in conflict"
end.

The above algorithm reduces the detection of conflict to the membership problem for fd's. Using Algorithm 3.1 also the membership problem for (fd's and) ad's can be reduced to that of fd's.

Algorithm 3.2. Membership of (fd's and) ad's

Input : A set $\mathcal{F}$ of fd's, a set $\mathcal{A}$ of ad's and an ad $X \not\to Y$. $\mathcal{F} \cup \mathcal{A}$ is assumed not to be in conflict.
Output : $\mathcal{F} \cup \mathcal{A} \models X \not\to Y$ or $\mathcal{F} \cup \mathcal{A} \not\models X \not\to Y$.
Method:
begin
if $\mathcal{F} \cup \{X \to Y\} \cup \mathcal{A}$ is in conflict
then return "$\mathcal{F} \cup \mathcal{A} \models X \not\to Y$"
else return "$\mathcal{F} \cup \mathcal{A} \not\models X \not\to Y$"
end.

Proof. If $\mathcal{F} \cup \mathcal{A} \models X \not\to Y$, then $\mathcal{F} \cup \{X \to Y\} \cup \mathcal{A}$ clearly is in conflict.

Conversely, let $\mathcal{F} \cup \{X \to Y\} \cup \mathcal{A}$ be in conflict (but $\mathcal{F} \cup \mathcal{A}$ not in conflict). Then Algorithm 3.1 finds an ad $T \not\to U \in \mathcal{A}$ for which $\mathcal{F} \cup \{X \to Y\} \models T \to U$. We prove that $\mathcal{F} \cup \{T \not\to U\} \models X \not\to Y$.

Let, for each set of attributes P,

$$\overline{P} = \{\text{attribute } A \mid \mathcal{F} \models P \to A\},$$

and let Ω be the set of all attributes. Suppose $\mathcal{F} \cup \{T \not\to U\} \not\models X \not\to Y$. There are two possible cases:

Case 1. $X \not\subset \overline{T}$. Consider the following instance R:

$\overline{T}$			$\Omega - \overline{T}$		
0	...	0	0	...	0
0	...	0	1	...	1

In R, $\mathcal{F}$ holds because of the definition of $\overline{T}$; $X \to Y$ holds since $X \not\subset \overline{T}$, and $T \not\to U$ holds since $U \not\subset \overline{T}$ (otherwise $\mathcal{F} \cup \mathcal{A}$ would have been in conflict). Hence $\mathcal{F} \cup \{X \to Y\} \not\models T \to U$, a contradiction.

Case 2. $X \subseteq \overline{T}$. It is easy to see that $U \not\subset \overline{\overline{YT}}$ because $T \not\to U$, $TY \to U$ and $T \to X$ would induce $X \not\to Y$, a contradiction. Consider the following instance R:

$\overline{YT}$			$\Omega - \overline{YT}$		
0	...	0	0	...	0
0	...	0	1	...	1

In R, $\mathcal{F}$ holds because of the definition of $\overline{YT}$, $X \to Y$ holds since $Y \subseteq \overline{YT}$, and $T \not\to U$ holds since $U \not\subset \overline{YT}$. Hence $\mathcal{F} \cup \{X \to Y\} \not\models T \to U$, a contradiction.

The fact that if $\mathcal{F} \cup \{T \not\to U\} \models X \not\to Y$, then also $\mathcal{F} \cup \mathcal{A} \models X \not\to Y$ completes the proof. □

4. The inheritance of dependencies

When performing a decomposition of a relation scheme, the question arises which dependencies hold in the subschemes. These dependencies are said to be *inherited* by the subschemes.

For the vertical decomposition the inheritance problem is quite trivial, but it may involve the calculation of all consequences of the given dependencies [1]. The inheritance of ad's, for the horizontal decomposition, is not trivial. Consider the following instance:

R:

A	B	C
0	0	0
0	1	0
1	0	1
1	1	1
1	0	0

In R, $A \not\to B$ and $B \not\to C$ hold. Let R be decomposed according to (A, C):

R_1:

A	B	C
0	0	0
0	1	0

R_2:

A	B	C
1	0	1
1	1	1
1	0	0

R_1 and R_2 did inherit A $\twoheadrightarrow$ B but not B $\twoheadrightarrow$ C.

To determine which dependencies are inherited by the subschemes one can use the following algorithm.

Algorithm 4.1. Decomposition according to a goal

Input : A scheme $\mathcal{R}(\mathcal{F}, \mathcal{A})$ and a goal (X, Y). $\mathcal{F} \cup \mathcal{A}$ is assumed not to be in conflict, $\mathcal{F} \not\models X \rightarrow Y$ and $\mathcal{F} \cup \mathcal{A} \not\models X \twoheadrightarrow Y$.

Output : A couple of schemes $(\mathcal{R}_1(\mathcal{F}_1, \mathcal{A}_1), \mathcal{R}_2(\mathcal{F}_2, \mathcal{A}_2))$, being the decomposition of $\mathcal{R}$ according to (X, Y).

Method:

var $\hat{\mathcal{A}}$: set of ad's := $\emptyset$;
begin
  for each T $\twoheadrightarrow$ U $\in \mathcal{A}$ **do**
  if $\mathcal{F} \models$ T $\rightarrow$ X
  then $\hat{\mathcal{A}}$:= $\hat{A} \cup$ {T $\twoheadrightarrow$ U};
  return $(\mathcal{R}_1(\mathcal{F} \cup \{X \rightarrow Y\}, \hat{A}),$
    $\mathcal{R}_2(\mathcal{F}, \hat{\mathcal{A}} \cup \{X \twoheadrightarrow Y\}))$
end.

This algorithm relies on [4, Theorems 5.3 and 5.11]. Note that the consequences of $\mathcal{F} \cup \mathcal{A}$ need not be calculated in order to obtain a generating set of dependencies for the subschemes.

5. The decomposition algorithm

From now on we let a relation scheme $\mathcal{R}$ have a set $\mathcal{G}$ of goals, as well as a set $\mathcal{F}$ of fd's, and $\mathcal{A}$ of ad's. When decomposing $\mathcal{R}$ according to (X, Y) $\in \mathcal{G}$ into $(\mathcal{R}_1(\mathcal{F}_1, \mathcal{A}_1, \mathcal{G}_1), \mathcal{R}_2(\mathcal{F}_2, \mathcal{A}_2, \mathcal{G}_2))$ we assume that for no goal (T, U) of $\mathcal{G}$ holds

$$\mathcal{F} \models T \rightarrow U \quad \text{or} \quad \mathcal{F} \cup \mathcal{A} \models T \twoheadrightarrow U,$$

and we let

$$\mathcal{G}_i = \{(T, U) \in \mathcal{G} \mid \mathcal{F}_i \not\models T \rightarrow U \text{ and } \mathcal{F}_i \cup \mathcal{A}_i \not\models T \twoheadrightarrow U\},$$
$$i = 1, 2.$$

The goals of $\mathcal{G}_i$ (i = 1, 2) are said to be *inherited* by $\mathcal{R}_i$.

A relation scheme $\mathcal{R}(\mathcal{F}, \mathcal{A}, \mathcal{G})$ is said to be in the *Inherited Normal Form* (INF) iff $\mathcal{G} = \emptyset$ or, for all (X, Y) $\in \mathcal{G}$, $\mathcal{F} \models$ T $\rightarrow$ U or $\mathcal{F} \cup \mathcal{A} \models$ T $\twoheadrightarrow$ U holds. A decomposition $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ is in INF iff all the $\mathcal{R}_i$ $(1 \leqslant i \leqslant n)$ are in INF.

Algorithm 5.1. Decomposition into INF

Input : A scheme $\mathcal{R}(\mathcal{F}, \mathcal{A}, \mathcal{G})$.

Output : If $\mathcal{F} \cup \mathcal{A}$ is not in conflict, a decomposition of $\mathcal{R}$ that is in INF.

Method:

if $\mathcal{F} \cup \mathcal{A}$ is in conflict
  then return an error message
  else delete all goals (X, Y) from $\mathcal{G}$ for which $\mathcal{F} \models X \rightarrow Y$ or $\mathcal{F} \cup \mathcal{A} \models X \twoheadrightarrow Y$;
decompose $\mathcal{R}(\mathcal{F}, \mathcal{A}, \mathcal{G}$ {after deletion}) using the procedure:

procedure decompose ($\mathcal{R}$: name, $\mathcal{F}$: set of fd's, $\mathcal{A}$: set of ad's, $\mathcal{G}$: set of goals)
begin
if $\mathcal{G} = \emptyset$
  then $\mathcal{R}$ is its own decomposition
  else decompose $(\mathcal{R}_1, \mathcal{F} \cup \{X \rightarrow Y\}, \hat{\mathcal{A}}, \mathcal{G}_1)$;
    decompose $(\mathcal{R}_2, \mathcal{F}, \hat{\mathcal{A}} \cup \{X \rightarrow Y\}, \mathcal{G}_2)$;
    the decomposition of $\mathcal{R}$ is
      (the decomposition of $\mathcal{R}_1$, the decomposition of $\mathcal{R}_2$)
end;

Since $\mathcal{G}_1$, $\mathcal{G}_2$ always contain at least one goal less than $\mathcal{G}$, Algorithm 5.1 stops after a finite time. From the definition of INF it is obvious that the final decomposition, produced by the algorithm, is in INF.

6. Example

Consider the following relation:

STAFF(E, J, S, M)
$\mathcal{F} = \{EJ \rightarrow SM\}$
$\mathcal{A} = \{M \twoheadrightarrow EJ\}$
$G = \{(E, J); (J, M), (E, M)\}$

A tuple (e, j, s, m) of an instance of STAFF means that employee e works on a job j, for which he (she) earns (only) one salary s and has (only) one manager m.

The ad M $\twoheadrightarrow$ EJ means that a manager must supervise more than one employee or job.

The fd's E $\rightarrow$ J, J $\rightarrow$ M and E $\rightarrow$ M do not hold. However, it is likely that most employees only

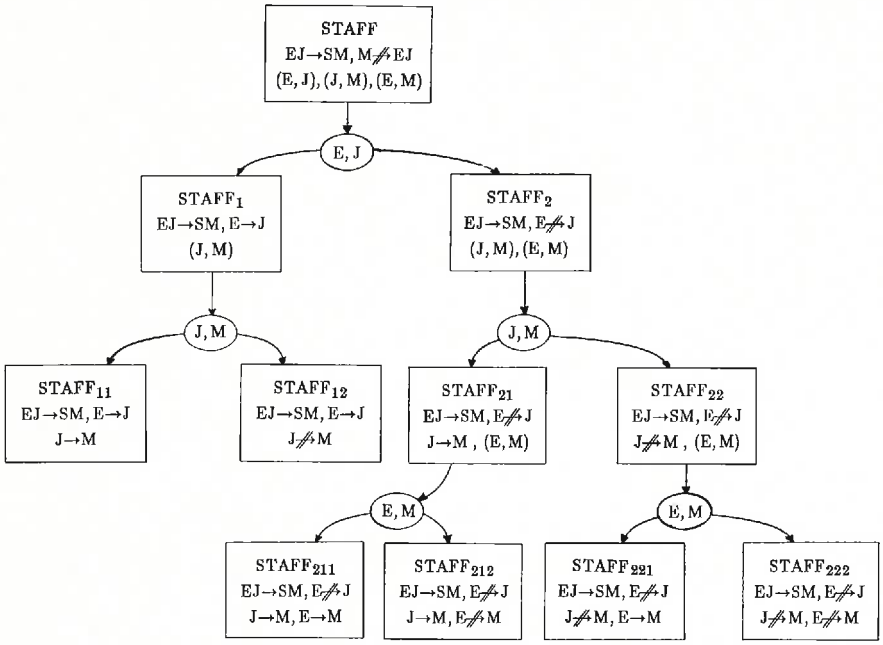

Fig. 1.

have one job, most jobs are supervised by only one manager, and hence most employees only have one manager. Using (E, J), (J, M) and (E, M) for horizontal decomposition leads to 6 subschemes, as is illustrated by the 'decomposition tree' of Fig. 1. The assumption about most employees and jobs means that the instances of STAFF_{11} will be considerably larger than the other subrelations. STAFF_{221} and STAFF_{222} should have the smallest instances.

From Fig. 1 one can easily see that the horizontal decomposition into INF preserves fd's but not all ad's. In [6] a dependency preserving decomposition algorithm is described. The algorithm of [6], however, does not always produce a subscheme (like STAFF_{11}) in which for all goals (X, Y) of $\mathcal{G}$ the fd $X \rightarrow Y$ holds.

7. Conclusion

We introduced an algorithm for the horizontal decomposition of a Relation which provides a mechanism for handling exceptions to functional dependencies.

This decomposition is 'functional dependency preserving', hence the traditional vertical decomposition can be used after the horizontal decomposition.

References

[1] C. Beeri and P.A. Bernstein, Computational problems related to the design of Normal Form relation schemes, ACM TODS 4(1) (1979) 30–59.

[2] P.A. Bernstein, Normalization and functional dependencies in the relational database model, CSRG-60 (1975).

[3] E. Codd, Further normalizations of the database relational model, in: R. Rustin, ed., Data Base Systems (Prentice-Hall, Englewood Cliffs, NJ, 1972) pp. 33–64.

[4] P. De Bra and J. Paredaens, The membership and the inheritance of functional and afunctional dependencies, Rept. 81–39, Dept. of Math., Univ. of Antwerp, Belgium, 1981.

[5] P. De Bra, Conditional dependencies, Thesis, Dept. of Math., Univ. of Antwerp, Belgium, 1981.

[6] J. Paredaens and P. De Bra, On horizontal decompositions, XP2-Congress, State Univ. of Pennsylvania, 1981.

[7] J. Ullman, Principles of Data Base Systems (Pitman, London, 1980).