数理解析研究所講究録
第363巻 1979年 130-164

LOGICAL DESIGN OF A 4NF D-TREE SCHEMA OF A RELATIONAL DATA BASE

YUZURU TANAKA

Graduate School of Information Engineering
Hokkaido University
Sapporo, 060 Japan

The design methodology of a sound schema that consistently reflects semantic structures of a real world is acquiring a greater importance to cope with both the increasing size and the versatile utilization of data bases. We show the design method to construct a BCNF and 4NF schema from the descriptions of the dependency structures of a data base. We include embedded MVD's as such dependency structures and analize them to reflect their structure as well as others in the synthesized schema. Our schema has an interrelational tree structure that increase the integrity and the handleability of a data base, and hence it is called a D-tree schema. With this interrelational structure, query processing can be highly automated. This paper reviews the BCNF D-tree schema with some significant modifications and extends this theory to the 4NF schema theory. Some heuristic considerations for the application order of FD's and MVD's to decompose a data base are also shown in this paper.

1. Introduction

The design methodology of a sound schema that consistently reflects semantic structures of a real world is acquiring a greater importance to cope with both the increasing size and the versatile utilization of data bases. Without the establishment of a method to describe and to analize semantic structures of information, it will become almost impossible in the near future to construct a sound schema.

It is the purpose of various studies on schema synthesis based on the relational framework [CODD70] to establish a theoretical basis necessary to cope with the difficulties of a sound schema synthesis [DELO72][ZANI76] [BERN76][FAGI77][TANA77].

In most of the researches, a schema is considered as an unstructured set of relations. In this view, a sound schema means an irredundant set of most simplified relations that represents a data base. The simplification means the separation of dependency structures to minimize the update operations including the validity check of dependencies. However, it is well known that the total separation of dependency structures can not always be achieved without abandonment of some dependencies to be embodied in a constituent relation of a synthesized schema [BEER78].

The synthesis approaches sacrifice the total separation, while the decomposition approaches sacrifice the embodiment of all dependencies and allow some of them to be treated as semantic constraints that are checked procedurally.

The interpretation of the representation and the irredundancy differs between these two kinds of approaches [BEER78]. The synthesis approaches strive for a minimum set of relations that embody all the dependency structures, while the decomposition approaches strive for minimal set of the most simplified relations that can represent the information content of a data base under design.

Our approach is classified among decomposition approaches. However, different from other approaches, we consider a schema as a structured set of relations. This is a very natural extension of a previously mentioned

view of a schema. This extension improves integrity of a schema by adding a global description of its dependency structures to a schema description. This global description enables us to remove both redundant relations from a schema and redundant attributes from each constituent relation. The removal of redundancy of this kind is necessary to guarantee the semantic integrity of a logical data base, and this is only possible in the decomposition approaches.

The tree structure is sufficient to describe such interrelational relationships. Thus a schema of this kind is named a dependency-tree schema, or simply a D-tree schema.

In our previous paper [TANA77], we introduced a D-tree schema consisting of Boyce-Codd normal form (BCNF) relations. Such a schema is called a BCNF D-tree schema. Fig.1 shows an example of a BCNF D-tree schema. We formalized an algorithm to construct a BCNF D-tree schema from a given set of functional dependencies (FD's) in a given set of attributes. If a synthesized D-tree schema is stored in a computer, the composition of a relational expression that computes a relation over an arbitrarily given subset of attributes can be automatically performed by a computer. This facility enables us to design both a highly nonprocedural query language and an automated mechanism for integrity checks on inconsistent update operations.

In this paper, we extend this D-tree schema theory to the fourth normal form (4NF) D-tree schema theory. The input set of dependencies is extended to include multivalued dependencies (MVD's) and embedded MVD's together with FD's. The review of the BCNF D-tree schema approach is also given with some improvements. As a common characteristic of decomposition approaches, the algorithm has nondeterministic aspects. Some heuristic consideration on this problem is also given in this paper.

2. Preliminaries

2.1. Relational model

A relation R(X) over an attribute set $X=\{A_1, A_2, \ldots, A_n\}$ is a subset of the cartesian product $Dom(A_1)\times Dom(A_2)\times\cdots\times Dom(A_n)$, where $Dom(A_i)$ is the domain of A_i. The projection of R(X) to a subset Y of X is $R(X)[Y]=\{\langle y\rangle \mid \langle y,z\rangle \in R(X)\}$, where Z=X-Y. The set of all the attributes of a data base Δ is denoted by Ω. In this paper, we assume that the uniqueness assumption holds in Ω, i.e., R(X)[Y]=R(Y) for any subsets $X \supset Y$ of Ω. This condition is not always satisfied. However, we believe Ω can be modified to satisfy this condition. The natural join of two relations R(X,Y) and S(Y,Z) is a relation $\{\langle x,y,z\rangle \mid \langle x,y\rangle \in R(X,Y)$ and $\langle y,z\rangle \in R(Y,Z)\}$ denoted by R*S, where X, Y, Z are disjoint. Since natural join is commutative and associative, the natural join of m relations $R_1, R_2, \ldots, R_m$ can be defined. This is denoted by $\Pi_{i=1}^{m} R_i$.

If each value of X in R(X,Y,Z) is associated with only one value of Y, it is said that there is a functional dependency (FD): X→Y. If $X,Y \subset Z$ and R(Z)=R(X,Y')*R(X,Y"), where Y'=Y-X and Y"=Z-X-Y, then it is said that there is a local multivalued dependency (LMVD): X↠Y in Z. The set Z is called a context. The definition of LMVD includes both MVD and embedded MVD (EMVD) defined by R. Fagin [FAGI77].

We conjecture the following set of axioms forms a complete set of axioms for FD's and LMVD's. Proofs of LMVD4 and 5 are shown in [TANA79]. It is well known that the set of axioms FD1-3, LMVD0-3, FD-LMVD1-2 forms a complete set of axioms for FD's and such LMVD's with a context equal to Ω [BEER77]. However, the completeness of the following axioms is unfortunately not proved yet.

FD1. (Reflexivity): If $Y \subseteq X$ then X→Y.

FD2. (Augmentation): If $Z \subseteq W$ and X→Y then XW→YZ.

FD3. (Transitivity): If X→Y and Y→Z then X→Z.

LMVD0. (Complementation): If X→Y in Z then X↠Z-Y in Z.

LMVD1. (Reflexivity): If $Y \subseteq X \subseteq Z$ then X↠Y in Z.

LMVD2. (Augmentation): If $V \subseteq W \subseteq Z$ and X↠Y in Z then XW↠YV in Z.

LMVD3. (Transitivity): If X↠Y in Z and Y↠W in Z then X↠W-Y in Z.

LMVD4. (Embedding): If $X \subseteq W \subseteq Z$ and X↠Y in Z then X↠$Y \cap W$ in W.

LMVD5. (Extension): If $X \twoheadrightarrow Y$ in Z and $(Z-Y) \twoheadrightarrow Y$ in W, where $W \supseteq Z$ then $X \twoheadrightarrow Y$ in W.

FD-LMVD1. If $X \rightarrow Y$ and $X, Y \subseteq Z$ then $X \twoheadrightarrow Y$ in Z.

FD-LMVD2. If $X \twoheadrightarrow Y$ in Z and $(Z-Y) \rightarrow Y$ then $X \rightarrow Y$.

2.2. Normal forms

As to the separation of dependency structures into a set of simplified dependency structures, various normal forms are proposed [CODD72][KENT73] [CODD74][BERN76][FAGI77][BEER78]. We define two of them that are important in this paper.

1. Boyce-Codd normal form (BCNF)

R(X) is in BCNF if, for any (Y, A) such that $Y \subset X$, $A \in X$, and $A \notin Y$, an FD: $Y \rightarrow A$ implies that $Y \rightarrow X$ holds.

2. Fourth normal form (4NF)

R(X) is in 4NF if a nontrivial LMVD: $Y \twoheadrightarrow Z$ in X implies an FD: $Y \rightarrow Z$.

The two main different kinds of approaches to the design of schemata take different normal forms as their basis. The synthesis approach takes 3NF that was first defined in [CODD72], and was modified in [BERN76], while the decomposition approach takes BCNF defined in [CODD74], or 4NF in [FAGI77].

2.3. Closure of dependencies.

To design a schema, it is necessary to know the closure of FD's and LMVD's, i.e., all dependencies inferable from given FD's and LMVD's. Let F and M denote respectively a given set of FD's and that of LMVD's. The closure is denoted by $(F, M)^{\dagger}$. Since the axioms LMVD3-5 are not convenient to use as inference rules, we introduce LMVD6 to replace them.

LMVD6. (LMVD interaction)

If $X \twoheadrightarrow Y$ in Z, $U \twoheadrightarrow V$ in W, $X \subset W$, and $U \subset Z$

then $X(Y \cap U) \twoheadrightarrow Y \cap V$ in $Z-(Y-W)$, and

$U(V \cap X) \twoheadrightarrow Y \cap V$ in $Z-(Y-W)$.

For the same reason, we replace FD-LMVD2 by FD-LMVD3.

FD-LMVD3. (FD-LMVD interaction)

If $X \rightarrow Y$, $U \twoheadrightarrow V$ in W and $X \subset W$ then $U(V \cap X) \rightarrow Y \cap V$.

The following Lemma holds with respect to these replacement.

Lemma 2.1.

A set of axioms FD1-3, LMVD0-5, and FD-LMVD1-2 is equivalent to a set of axioms FD1-3, LMVD0-2, LMVD6, FD-LMVD1, and FD-LMVD3 [TANA79].

We denote FD part and LMVD part of $(F, M)^{\dagger}$ by $(F, M)^{\dagger}_{FD}$ and $(F, M)^{\dagger}_{LMVD}$ respectively. The partial closure $(F, M)^{I\dagger}$ is a set of all FD's and all LMVD's inferable from (F, M) by a subset I of a set of inference rules given in section 2.1.

The following Lemma assures the computability of $(F, M)^{\dagger}_{FD}$ without considering LMVD-LMVD interaction.

Lemma 2.2.

Let I be a set of axioms {FD1-3, LMVD0, FD-LMVD3}, i.e., I does not include LMVD1-2, LMVD6 and FD-LMVD1. Then the following holds true.

$$(F, M)^{I\dagger}_{FD} = (F, M)^{\dagger}_{FD}$$ [TANA79].

3. Schema of a data base

3.1. Decomposition vs. synthesis

A pair (Ω, Γ), where Ω is an attribute set of a data base Δ and Γ is a dependency structure, is called a scheme over Ω. A schema is a set of schemes over subsets of Ω, i.e., $\{(\Omega_i, \Gamma_i)\}_{i=1}^{k}$, that represents (Ω, Γ) with least redundancy. The definitions of representation and redundancy differ among researchers. In synthesis approaches, representation means that

$$\Gamma^{\dagger} = (\bigcup_{1 \leq i \leq k} \Gamma_i)^{\dagger},$$

and irredundancy means that

$$\forall i, \nexists f \in \Gamma_i \text{ s.t. } \Gamma^{\dagger} = ((\bigcup_{\substack{1 \leq j \leq l \\ j \neq i}} \Gamma_j) \cup (\Gamma_i - \{f\}))^{\dagger}.$$

While, in decomposition approaches, representation and irredundancy repectively means that

1. (representation)
$$R(\Omega) = \prod_{1 \leq i \leq k} R(\Omega_i)$$

and

2. (irredundancy)
$$\forall i, \nexists A \in \Omega_i \text{ s.t. } R(\Omega) = (\prod_{\substack{1 \leq j \leq k \\ j \neq i}} R(\Omega_j)) * R(\Omega_i - \{A\})$$

[BEER78].

It is well known that it is not always possible to make each (Ω_i, Γ_i) in BCNF in synthesis approaches, nor to make a schema to embody all dependency structures, i.e., to make $\Gamma^{\dagger} = (\bigcup_{1 \leq i \leq k} \Gamma_i)^{\dagger}$, in decomposition approaches.

Our approach stands on a basis of decomposition approaches, however, some part of our ideas presented in this paper may be also applicable to synthesis approaches.

Since the equivalence of $\Gamma^{\dagger}$ and $(\bigcup_{i=1}^{k} \Gamma_i)^{\dagger}$ does not hold in decomposition approaches, a minimal set Γ_r of dependencies satisfying $\Gamma^{\dagger} = ((\bigcup_{i=1}^{k} \Gamma_i) \cup \Gamma_r)^{\dagger}$ should be considered in decomposition approaches. We call this set a residue dependency structure. This set Γ_r plays an important role in procedural validity checks upon inconsistent updates.

Therefore a schema is characterized by $(\{(\Omega_i, \Gamma_i)\}, \Gamma_r)$ in decomposition approaches.

3.2. Dependency-tree schema

Some part of the dependency structure Γ defines interrelational dependency structure among $\{(\Omega_i, \Gamma_i)\}$. It is desirable for a schema to have a description about this structure since time invariant structures of a data base should be reflected in a schema to increase integrity and handleability of a schema. The interrelational dependency structure defines a graph structure G among $\{(\Omega_i, \Gamma_i)\}$. Therefore a schema is characterized by $(\{(\Omega_i, \Gamma_i)\}: G: \Gamma_r)$. However, it is sufficient to define a tree structure T instead of a general graph G among $\{(\Omega_i, \Gamma_i)\}$ since, under the uniqueness assumption, multiple associations between two constituent schemes represent a same semantic relationship between them, and hence superfluous.

Therefore, a schema denotes an irredundant set of schemes $(\{(\Omega_i, \Gamma_i)\}; T; \Gamma_r)$ that represents (Ω, Γ). A schema of this type is called a dependency-tree schema, or a D-tree schema. A D-tree schema with every (Ω_i, Γ_i) in X-normal form is called a X-NF D-tree schema. Fig.1 in chapter 1 shows an example of a BCNF D-tree schema.

3.3. Dependency-diagram

In this section, we only consider FD's, i.e., $\Gamma=F$. For each FD f: X-→Y, let $\varepsilon(f)$ denote the set of all the attributes that are functionally dependent on X. This set is called the maximum dependent set of f. It should be noticed that $\varepsilon(f)$ includes X. For each f in F, $\varepsilon(f)$ can be calculated using FD1-3. For an attribute set Z, a subset W of Z is called a determinant of Z if W-→Z holds and no subsets of W satisfy this.

For an FD f:X-→Y, X is a candidate of determinants of $\varepsilon(f)$. If $\varepsilon(f) = \varepsilon(g)$ holds for f:X-→Y and g:U-→V, X and U are both candidates of determinants of $\varepsilon(f)$. If both $\varepsilon(f)=\varepsilon(g)$ and $X \subset U$ hold then U is superfluous as a candidate since X is a stronger candidate than U. We define a list of candidate determinants of $\varepsilon(f)$ as a family of minimal attribute sets each of which determines $\varepsilon(f)$. If $\varepsilon(f) \supset \varepsilon(g)$ holds then

the FD:X→U holds. In other words, a candidate determinant of ε(f) determines every candidate determinant of ε(g).

We define an equivalence relation f≡g in F as

$$f \equiv g \quad \text{iff} \quad \varepsilon(f)=\varepsilon(g).$$

The classification of F by ε is a set of equivalence classes with respect to "≡". This is denoted by F/ε. For each g in F/ε, the maximum dependent set ε(g) and the list τ(g) of candidate determinants are defined as

$$\varepsilon(g)=\varepsilon(f) \quad (\ g=[f]\ ),$$

and

$$\tau(g)=\ker(\cup_{f\in g}\ \{\text{left}(f)\}),$$

where left(f) denotes the attribute set on the left of f, and ker is defined for a family S of sets as

$$\ker(S)=\{s \mid s\in S, \nexists s'\in S(s \supset s')\}.$$

We define a partial order ">" in F/ε as

$$f>g \quad \text{iff} \quad \varepsilon(f) \supset \varepsilon(g).$$

In the sequel, we assume F includes a special trivial FD $f_0:\Omega\rightarrow\phi$. The equivalence class including f_0 is denoted by g_0. The partial order ">" uniquely defines a Hasse diagram $(F/\varepsilon, >)$ for a given scheme (Ω, F) of a data base Δ. A Hasse diagram with values of ε and τ at each node is called a dependency-diagram, or a D-diagram of Δ and is denoted by $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$. It should be noticed that a D-diagram $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$ is a compiled view of (Ω, F) and that $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$ is equivalent to (Ω, F) in a sense that $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$ has the same dependency structures as (Ω, F).

In Fig.2, we show an example set of FD's. Fig.3 shows their maximum dependent sets, and Fig.4 its D-diagram.

It may probably be the best way irrespective of the destination normal form to start schema synthesis with the D-diagram $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$, since it clearly describes the overall dependency structure of Δ.

4. BCNF D-tree schema

4.1. Definition of a BCNF D-tree schema

In this section, only FD's are considered. Thus a D-tree schema is a tuple $(\{(\Omega_i, F_i)\}; T; F_r)$ equivalent to (Ω, F). A D-tree schema is said to be a BCNF D-tree schema if every scheme (Ω_i, F_i) is in BCNF. Since (Ω_i, F_i) is in BCNF, a list of determinants of Ω_i is sufficient to describe its dependency structure. Let N be a finite set, $attr(n_i)$ be Ω_i for n_i in N, and $\tau(n_i)$ be the list of determinants of Ω_i. A tree structure T is represented by a triple (N, par, n_0), where n_0 is a special element of N called a root and par is a function that defines for each node its parent node in T, i.e., $par: N-\{n_0\} \to N$. For each node n in N, we choose one determinant denoted by $\alpha(n)$ as a key of this node. Then a BCNF D-tree schema can be characterized by an octuple $(N, n_0, attr, \alpha, \tau, par, \Omega; F_r)$.

The conditions for an octuple $(N, n_0, attr, \alpha, \tau, par, \Omega; F_r)$ to be a BCNF D-tree schema are listed below.

(1) (representability)

$$R(\Omega)=\Pi_{n\in N} R(attr(n))$$

(2) (irredundancy)

$$\forall_{n\in N}, \nexists_{A\in attr(n)} \left(R(\Omega)=(\Pi_{n'\in N-\{n\}} R(attr(n')))*R(attr(n)-\{A\}) \right)$$

(3) (existence of a key)

$$\forall_{n\in N} \left(\alpha(n)\neq\phi \text{ and } \alpha(n)\to attr(n) \right)$$

(4) (existence of dependants)

$$\forall_{n\in N-\{n_0\}} \left(attr(n)\neq\alpha(n) \right)$$

(5) (extended Boyce-Codd property)

$$\neg\exists n \in N \left(\exists X \subset \Omega, \exists A\in attr(n)-X \text{ s.t. } \alpha(n)\to X, X\to A, \text{ and } A \text{ is nonprime in the set of all the attributes dependent on } X \right)$$

(6) (downward interrelational dependency)

$$\forall_{n\in N-\{n_0\}} \left(\alpha(par(n))\to\alpha(n) \right)$$

(7) (a list of determinants)

$$\forall_{n\in N} \left(\tau(n)=ker(\tau(n)),\ \tau(n) \subset 2^{attr(n)}, \text{ and } \forall_{X\in\tau(n)} \left(X\to\alpha(n) \right) \right)$$

(8) (residue FD's)

F_r is a minimal set of FD's satisfying

$$((\cup_{n\in N} \cup_{X\in T(n)} \{X\rightarrow attr(n)\}) \cup F_r)^{+} = F^{+}.$$

The condition 5 differs from BC property described in the definition of BCNF in section 2.2. Since BC property is defined within an attribute set, even a set of two subschemes ({A,C}, {A→C}) and ({A,B}, {A→B}) for a scheme ({A,B,C}, {A→B, B→C}) are both in BCNF. However, this separation is not desirable. To guarantee such a desirable separation as ({A,B}, {A→B}) and ({B,C}, {B→C}), BC property must be replaced by extended BC property.

4.2. Algorithm for BCNF D-tree schema design

Our algorithm uses the D-diagram of Δ as a basis of schema design. Let $(F/\varepsilon, >, \varepsilon, \tau, \Omega)$ be the D-diagram of Δ. If a node $g\in F/\varepsilon$ has no son nodes, $R(\varepsilon(g))$ is already in BCNF. Let attr(g) be defined as $\varepsilon(g)$ for such a node g. The key of g is arbitrarily chosen among members of $\tau(g)$. Suppose that a node g has son nodes $g_1, g_2, \ldots, g_h$ and that the values of attr and α at these son nodes are already calculated. We define the list $\omega(g)$ of attribute sets as a set:

$$\{X \mid X\subset\varepsilon(f),\ X\rightarrow\varepsilon(f),\ R(X) \text{ is in BCNF, and } \nexists Y\supset X\ (\ Y\subset\varepsilon(f),\ Y\rightarrow X, \text{ and } R(Y) \text{ is in BCNF }\}.$$

The following program calculates $\omega(g)$.

```
procedure ω(g);
  begin
    ω:={ε(g)};
    while there exists g_i for some σ in ω s.t. σ∩(ε(g_i)-α(g_i))≠φ
    do
      begin
        ω:=ker(ω∪{(σ-ε(g_i))∪α(g_i)};
      end;
  end;
```

Let attr(g) be an arbitrarily chosen attribute set in $\omega(g)$. Then R(attr(g)) has no transitive dependency and thus it is in BCNF.

Corresponding to attr(g), the list $\tau(g)$ of candidate determinants of $\varepsilon(g)$ should be modified so that each element of $\tau(g)$ becomes a candidate determinant of attr(g). In this process, some determinants of $\varepsilon(g)$ are found impossible to be embodied by R(attr(g)). These should be listed in F_r without redundancy. However, we do not show the computation process of F_r in this paper. This is explained in the previous paper [TANA77], and the utilization of F_r for integrity checks will be described elsewhere.

```
procedure τ(g);
  begin
    while there exists a son node g_i of g for some σ s.t.
    σ ⊄ attr(g) and σ ∩((ε(g_i)-α(g_i))≠φ
    do
      begin
        τ(g):=ker((τ(g)-{σ}) ∪ {α(g_i) ∪ (σ-ε(g_i))});
      end;
    α(g):=an arbitrary element of τ(g);
  end;
```

The computation process of attr, α, and τ for the D-diagram in Fig.4 is shown in Fig.5. The resultant diagram is a kind of schemata, however, its irredundancy is not guaranteed.

Let Mark(r, X) and Delete(S, X) be two procedures defined below, where ID(r) denotes a set of all the son nodes of r in F/ε. We assume that, for each g, $\mu(g)$ is initially equal to ϕ.

```
procedure Mark(r, X);
  begin
    μ(r):=X;
    Y:=X;
    X:=X∩(ε(r)-α(r));
    S:=ID(r);
    while there exists r'∈S do
      begin
        Z:=Y;
        S:=S-{r'};
        Mark(r', Z);
```

```
        Y:=Y-Z;
        Delete(S, Z);
      end;
  end;

procedure Delete(S, X);
  begin
    S':=S;
    while there exists r'εS' do
      begin
        S':=S'-{r'};
        attr(r'):=α(r') ∪ (μ(r') ∩attr(r')) ∪ (attr(r')-X);
        T:=ID(r');
        Delete(T, X);
      end;
  end;
```

The execution of Mark(g_0, Ω) will remove all redundant attributes from the previously obtained redundant schema. This execution for Fig.5 (d) is shown in Fig.6.

The par function of a BCNF D-tree schema is defined as follows by truncating superfluous links in the diagram.

par(g)=if attr(g')≠α(g') or g'=g_0 then g'
else par(g'), where g' is a parent node of g satisfying that μ(g')⊃ μ(g).

The irredundant BCNF D-tree schema can be obtained by deleting those nodes except g_0 that have no dependant part, i.e., α(g)=attr(g). The conditions (3),(5),(6) in section 4.1 holds from the definition of par, (1) and (2) holds from the definition of μ, and (8) holds from the construction method of F_r. Fig.7 shows a resultant BCNF D-tree schema of the example data base in Fig.2.

Since the algorithm is nondeterministic at several steps, man-machine interaction may be necessary to design a better schema. Otherwise, we must define some cost function for each nondeterministic step listed below:

1. the selection of attr(g) out of $\omega(g)$,
2. the selection of $\alpha(g)$ out of $\tau(g)$,
3. the order of marking process, i.e., the calculations of $\mu(g)$'s, in Mark(g_0, Ω).

4.3. Query Processing with a BCNF D-tree schema

A query has a following general form under the uniqueness assumption:

SELECT Q_1
WHERE Pred(Q_2; $\mathbf{c}$),

where Q_1 and Q_2 are attribute sets, Pred is a predicate, and $\mathbf{c}$ is a vector constant. The relational expression for this query is expressed as

$$R(Q_1 \cup Q_2)[\text{Pred}(Q_2; \mathbf{c})][Q_1],$$

where [Pred(Q_2; $\mathbf{c}$)] is a restriction and [Q_1] denotes projection. It is not recommended to execute this expression directly. However, some optimization techniques are already known [SMIT75]. Therefore, if we can automate the process to compose a relational expression for $R(Q_1 \cup Q_2)$, we can design a highly nonprocedural query language.

This section shows an algorithm to compose the most simplified relational expression for R(Q), where Q is an arbitrarily given subset of Ω. Let (N, n_0, attr, α, τ, par, Ω) be a BCNF D-tree schema. For each A in Ω, define the uniquely determined node node(A) in N as

$$\text{node}(A)=\text{if } A \in \text{attr}(g_0) \text{ then } g_0 \text{ else } g \text{ s.t. } A \in \text{attr}(g)-\alpha(g).$$

Let des(g) denote all the descendant nodes of g including g itself, and Rel(g) denote R(attr(g)). We define a minimal node in a subset N' of N as a minimal element of N' with respect to the partial order ">". A relational expression for R(Q) is given by rel(Q) of the following procedure.

```
procedure rel(Q);
  begin
    rel:= 𝟙;  ( 𝟙 is the identity of join operation, i.e., 𝟙=R(ϕ).)
    M:=N;
    while Q≠ϕ do
```

```
begin
  Ω_d:=∪_{r∈M} (α(r)-μ(r));
  if Q-Ω_d=φ
    then P:=Q
    else P:=Q-Ω_d;
  r:=minimal node of {node(A) | A∈P};
  if Q-attr(r)=φ
   then
     begin
       rel:=rel*(Rel(r)[Q]);
       Q:=φ;
     end
    else
     begin
       rel:=rel*(Rel(r)[α(r)∪Q]);
       Q:=(Q-attr(r))∪α(r);
       M:=M-des(r);
     end;
end;
end;
```

In Fig.8, the composition process of a relational expression for Q={I, J} in the example data base Fig.7. is shown.

5. 4NF D-tree schema

5.1. Some considerations on 4NF D-tree schema synthesis

The purpose of this chapter is to extend the BCNF D-tree schema theory to the 4NF theory. Therefore, in this chapter, the dependency structure is a pair (F, M), where F, M are sets of FD's and LMVD's respectively.

The calculation of $(F, M)^{\dagger}$ by the set of axioms given in section 2.3 is not a difficult task. However, we have to consider the appropriate order in which we apply each LMVD to decompose the data base. One might suppose that decomposition by FD's should precede that by LMVD's or vice versa. However, that is not the case as shown in Fig.9 and Fig.10. In Fig.9, priority given to FD's cause redundancy, while, in Fig.10, opposite situation occurs.

Fig.11 shows two alternative decomposition order between two MVD's. The redundancy occurs when the LMVD with a left attribute set including that of the other is applied first to decompose the data base.

Further observation indicates that the redundancy may occur if the decomposition by an LMVD f precedes that by another g such that the left attribute set of g is functionally dependent on that of f. The previous examples in Fig.9 and Fig.10 are such cases.

Besides, the following heuristic observation also approves the criterion in which the decomposition by an LMVD f should not precede the decomposition by g such that the left attribute set of g is functionally dependent on that of f. Fig.13 shows two alternative decomposition orders for a data base in Fig.12, where Fig.13 (a) contradicts the criterion. Although it causes no redundancy in this case, the table corresponding to {<Employee>, <Item>, <Supplier>} in Fig.13 (a) will become a very large table comparing to {<Department>, <Item>, <Supplier>} in Fig.13 (b). This is a different type of redundancy.

5.2. Closure of FD's and LMVD's

For the convenience of computation of closures, we introduce a standard representation of an LMVD. The standard representation of an LMVD f with a context Z has a form:

$$X : [\ Y_0\ ]\ \ Y_1 \mid Y_2 \mid \cdots \mid Y_n,$$

where $\{X, Y_0, Y_1, \ldots, Y_n\}$ is a partition of Z, $X \rightarrow Y_0$ and $X \twoheadrightarrow Y_i$ in Z for any i. Three functions are defined for a standard representation;

$$\text{context}(f) = Z,$$
$$\text{left}(f) = X,$$
$$\text{right}_i(f) = \text{if } i>n \text{ then } \phi \text{ else } Y_i.$$

In the sequel, we assume that LMVD's are in standard form.

Let F' and M' denote the FD part and LMVD part of $(F, M)^{I\dagger}$, where I ={FD1-3, LMVD0, FD-LMVD3}. By Lemma 2.3, $F'=(F, M)^{\dagger}_{FD}$ holds. If M is initially represented in standard form, LMVD0 can be deleted from inference rules I. To obtain the intermediate set M', we have to remove FD's from M. For each f in M, if there exists a functional dependency from a subset X of left(f) to all the attributes in left(f), left(f) should be replaced by X and left(f)-X should be moved from left(f) to $\text{right}_0(f)$. All attributes in $\text{right}_i(f)$ that are functionally dependent on left(f) should be also moved from $\text{right}_i(f)$ to $\text{right}_0(f)$. Then, for each functional dependency f: X -→ Y satisfying that Y is a maximum set dependent on X and X is a minimal set that determines $X \cup Y$, we add an LMVD g to M' that is defined as

$$\text{context}(g) = \Omega,$$
$$\text{left}(g) = X,$$
$$\text{right}_0(g) = Y,$$
$$\text{right}_1(g) = \Omega\text{-}X\text{-}Y,$$
$$\text{right}_i(g) = \phi \quad (\ i>1\ ).$$

The closure of all LMVD's is obtained as the LMVD-closure of M', i.e., $(M')^{I''\dagger}$, where I''={LMVD0-2, LMVD6}. Instead of calculating all LMVD's in $(M')^{I''\dagger}$, we calculate a set M'' of LMVD's that satisfies the following conditions.

$\forall_{f \in (M')^{I"\dagger}}$, $\exists 1_{g \in M"}$ such that

1. (Minimality of left(g))
   left(g) $\subset$ left(f),
2. (Maximality of $right_0$(g))
   $right_0(g) \supset right_0(f)$,
3. (Minimality of left(g) $\cup$ $right_j$(g) ($j \neq 0$))
   $\forall_{i \neq 0}$, $\exists_{j \neq 0}$ s.t.
   left(g) $\cup$ $right_j(g) \subset$ left(f) $\cup$ $right_i(f)$,
4. (Maximality of context(g))
   context(g) $\supset$ context(f).

Since the calculation of $M"$ is the direct application of LMVD6 on M' in standard representation with some modification to the resultant LMVD's by LMVD0-2, we do not show the details of this process here.

Fig.14 shows example sets of F and M. The FD-closure F' and the intermediate set M' of LMVD's are calculated in Fig.15. The LMVD-closure $M"$ of M' is shown in Fig.16.

5.3. 4NF decomposition

We now show how to use (F', $M"$) to decompose a data base into 4NF relations. Since the completeness of the set of axioms in section 2.1 is not proved yet, we can not say that the following procedures produce a 4NF D-tree schema. However, since our algorithm is based on the closure of dependencies obtained by the axioms in section 2.1, a D-tree schema obtained by this algorithm is assured of its 4NF property if the axioms are proved to be complete.

Let $M*$ denote a standard representation of F' and $M"$. The execution of Decomp(Ω, $M*$) will produce a 4NF D-tree schema following the strategy considered in section 5.1. The procedure Decomp(X, M) is defined as follows.

```
procedure Decomp(X, M);
  begin
    if there exists f ε M s.t.
      1. context(f) ⊃ X, left(f) ⊂ X,
      2. the set {i_1, i_2, ..., i_k} of all integers s.t.
         right_ij(f) ∩ X ≠ φ has more than two integers,
      3. there is no other such LMVD g as above in M that also
         satisfies a functional dependency left(f) → left(g),
    then
      Decomp:=Π_{1≤j≤k}(Decomp(left(f) ∪ (right_ij(f) ∩ X), M))
    else
      Decomp:=R(X);
  end;
```

The relational expression Decomp(Ω, M*) with parentheses defines a decomposition tree. Fig.17 (a) shows M* of a data base in Fig.14. Fig.17 (b) and (c) respectively show the dependency diagram among left attribute sets of dependencies in M* and a 4NF D-tree schema obtained by the execution of Decomp(Ω, M*).

We now prove the weak irredundancy of this decomposition.

Theorem 5.1.

For any constituent relation R_{k_i} of a 4NF D-tree schema Decomp(Ω, M"), there exists no such subexpression E of Decomp(Ω, M") satisfying that $E=\Pi_i E_i$, $E_k=\Pi_{j\ j\neq i} E_{k_j} * R_{k_i}$, and R_{k_i} is a projection of either $\Pi_{i\neq k} E_i$ or $\Pi_{j\neq i} E_{k_j}$.

proof

Fig.18 shows this situation. Let Ω_k, Ω_k*, Ω_{ki}, Ω_{ki}* denote respectively the sets of attributes in E_k, $\Pi_{j\neq k} E_j$, R_{k_i}, $\Pi_{j\neq i} E_{k_j}$. Then it holds that $E=R(\Omega_k*)*R(\Omega_{ki}*)*R(\Omega_{ki})$. If $\Omega_{ki}\subset\Omega_k$* holds, it holds that $E=R(\Omega_k*)*R(\Omega_{ki}*)$. Thus $\Omega_k*\cap\Omega_{ki}*\twoheadrightarrow\Omega_k*|\Omega_{ki}*$ in the set of attributes in E. Since,for any j, the intersection of the attribute set of E_j and Ω_k is equal to the intersection of

all the attribute sets of E_j's in decomposition trees, it holds that the left attribute set of the LMVD that was used to get the sub-decomposition-tree E is equal to $\Omega_k^* \cap \Omega_k$. If $\Omega_k^* \cap \Omega_{ki}^* \subset \Omega_k^* \cap \Omega_k$ holds, it contradicts the minimality of $\Omega_k^* \cap \Omega_k$. Thus $\Omega_k^* \cap \Omega_{ki}$ should be equal to $\Omega_k \cap \Omega_k^*$, which is equal to $\Omega_{ki}^* \cap (\Omega_{ki}\Omega_{ki}^*)$. Hence it holds that $\Omega_{ki} \subset \Omega_{ki}^*$. Since it holds that Ω_{ki} is a subset of $\Omega_{ki} \cap \Omega_{ki}^*$, R_{k_i} could not be a constituent relation. This contradicts the assumption.

The query processing with 4NF D-tree schema is similar to that with BCNF D-tree schema. However, the irredundancy of an expression is not guaranteed unless LMVD's hidden in this schema are taken into consideration. This is illustrated in Fig.19.

6. Concluding remarks

The D-tree schema approach presented in this paper may be usefull to cope with the logical problems of data bases concerning data base integrity. We are implementing a relational query language based on this approach that is similar to SEQUEL but has default facility to analyze the access path when FROM clause is omitted. The validity check of update operation can be also treated following this approach. We can find a set of relations necessary to check dependency structures with respect to the validity of update request. It should be noticed that the introduction of interrelational tree structure is not the complication of a schema but the simplification of it. Therefore, the handleability of a schema is increased significantly.

On the other hand, this simplification may cause various problems concerning a gap between the reality and its rather mathematical model [KENT78]. Further studies such as [SCHM75][SMIT77] may be necessary to fill up this gap. One of them may be a theoretical basis for a design of attribute sets each of which satisfies the uniqueness assumption.

Problems also exist among different interpretation of mathematical models. Although the concept of independent relations by J. Rissanen [RISS77] has suggested some theoretical orientation, we do not think this interpretation problem like the selection problem among synthesis and decomposition approaches may be tractable in mathematical formalizations. We believe some of the integrity checks must be done procedurally. Hence we think it is not absolutely necessary to make a schema to embody all dependencies. This is the reason we are standing on the basis of decomposition approaches.

As to the decomposability of a relation, the concept of mutual dependency [NICO78] may be interesting to consider in D-tree framework. However, much more studies may be necessary to know complete mathematical properties of relational structures.

References

[BEER77] C. Beeri, R. Fagin, and J.H. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies," Proc. ACM-SIGMOD Conf., Toronto, Aug. 1977, pp. 47-61.

[BEER78] C. Beeri, P.A. Bernstein, and N. Goodman, "A Sophisticate's Introduction to Database Normalization Theory," Proc. 4th VLDB Conf., Berlin, Sept. 1978, pp. 113-124.

[BERN76] P.A. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM Trans. on Database Sys.*, vol. 1, No. 4 (Dec. 1976), pp. 277-298.

[CODD70] E.F. Codd, "A Relational Model for Large Shared Data Bases," *CACM*, Vol. 13, No. 6 (June, 1970), pp. 377-387.

[CODD72] E.F. Codd, "Further Normalization of the Data Base Relational Model," in *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 33-64.

[CODD74] E.F. Codd, "Recent Investigations in Relational Data Base Systems," Proc. IFIP 74, North-Holland, 1974, pp. 1017-1021.

[DELO72] C. Delobel and R.C. Casey, "Decomposition of a Data Base and the Theory of Boolean Switching Functions," *IBM J. of Res. and Dev.*, Vol. 17, No. 5 (Sept. 1972), pp. 370-386.

[FAGI77] R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Data Bases," *ACM Trans. on Database Sys.*, Vol. 2, No. 3 (Sept. 1977), pp. 262-278.

[KENT73] W. Kent, "A Primer of Normal Forms," IBM Sys. Dev. Div., TR02.600, San Jose, Cal., 1973.

[KENT78] W. Kent, *Data and Reality*, North-Holland, Amsterdam, 1978.

[NICO78] J.M. Nicolas, "Mutual Dependencies and Some Results on Undecomposable Relations," Proc. 4th VLDB Conf., Berlin, Sept. 1978, pp. 360-367.

[RISS77] J. Rissanen, "Independent Components of Relations," *ACM Trans. on Database Sys.*, Vol. 2, No. 4 (Dec. 1977), pp. 317-325.

[SCHM75] H.A. Schmid and J.R. Swenson, "On the Semantics of the Relational Data Model," Proc. 1975 ACM-SIGMOD Conf., San Jose, Cal., pp. 211-223.

[SMIT75] J.M. Smith and P.Y.T. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," *CACM*, Vol. 18, No. 10 (Oct. 1975), pp. 568-579.

[SMIT77] J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. on Database Sys.*, Vol. 2, NO. 2 (June 1977), pp. 105-133.

[TANA77] Y. Tanaka and T. Tsuda, "Decomposition and Composition of a Relational Database," Proc. 3rd VLDB Conf., Tokyo, Oct. 1977, pp. 454-461.

[TANA79] Y. Tanaka, "The Design of a 4NF D-tree schema," Tech. Rep. HU-ENZ-7900, (Available from the author).

[ZANI76] C. Zaniolo, "Analysis and Design of Relational Schemata for Database Systems," Tech. Rep. UCLA-ENG-7769, Dept. of Computer Science, UCLA, July 1976.

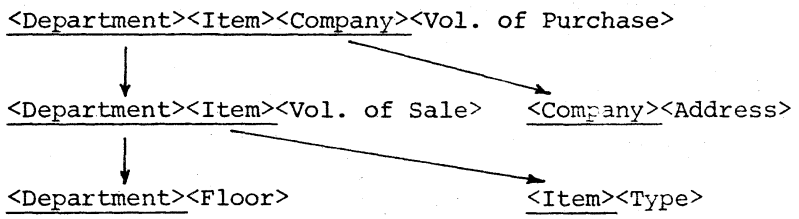


Fig. 1 An example of a BCNF D-tree schema.
(A data base of a department store)

F:

f_1 : A B C --→ E
f_2 : A E --→ C B
f_3 : B --→ D H
f_4 : D --→ E
f_5 : G --→ D F J
f_6 : D F --→ I
f_7 : H --→ B

Fig. 2 An example of a set of functional dependencies.

ε:

$\varepsilon(f_0)$ = {A, B, C, D, E, F, G, H, I, J}
$\varepsilon(f_1)$ = {A, B, C, D, E, H}
$\varepsilon(f_2)$ = {A, B, C, D, E, H}
$\varepsilon(f_3)$ = {B, D, E, H}
$\varepsilon(f_4)$ = {D, E}
$\varepsilon(f_5)$ = {D, E, F, G, I, J}
$\varepsilon(f_6)$ = {D, E, F, I}
$\varepsilon(f_7)$ = {B, D, E, H}

Fig. 3 Maximum dependent sets of functional dependencies in Fig.2.

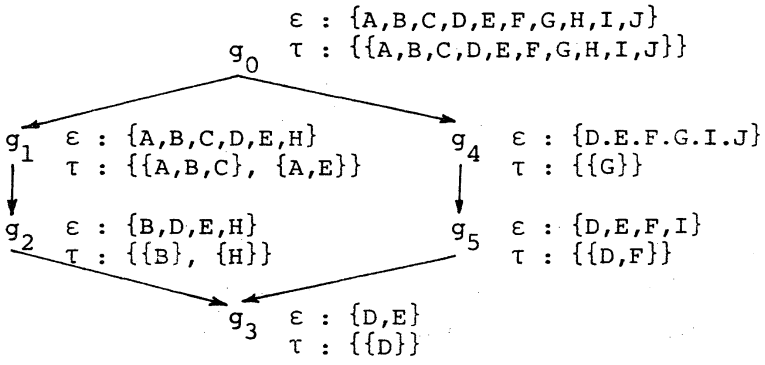


Fig. 4 D-diagram of a data base in Fig.2.

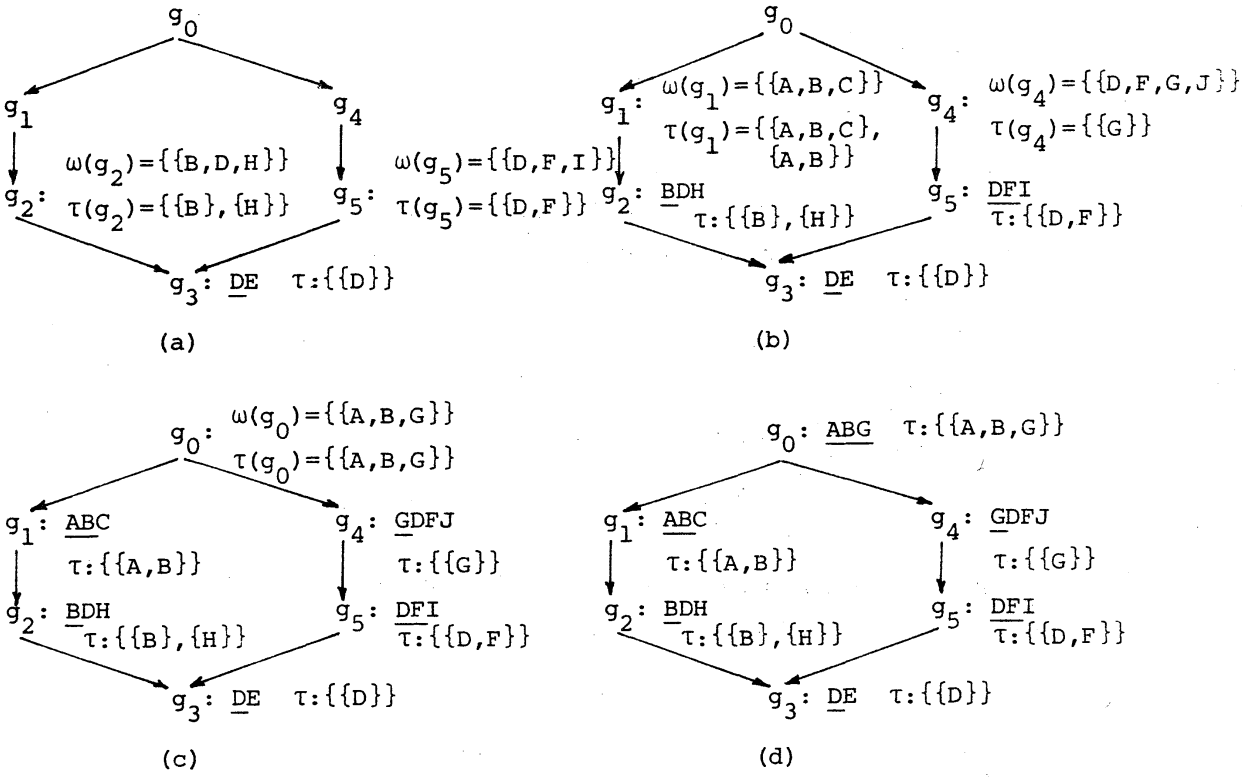
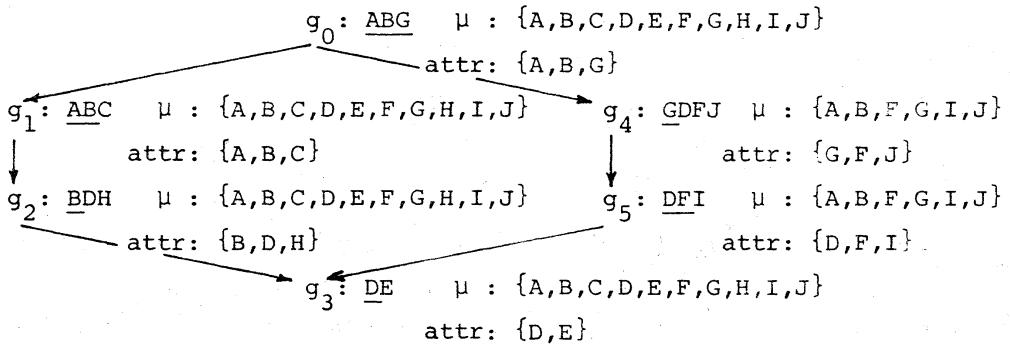


Fig. 5 Computation of attr, α, and τ of a data base in Fig.2.



The attribute "D" in g_4 is redundant.

Fig. 6 The removal of redundancy from a schema by the marking process Mark(g_0, Ω).

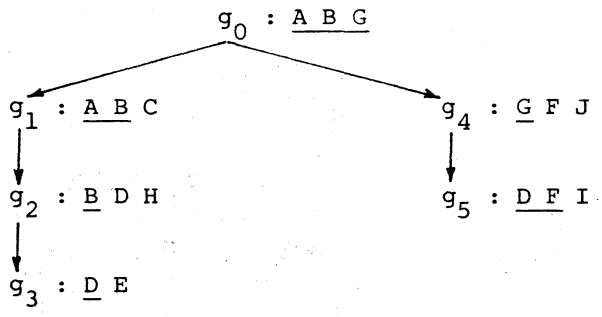


Fig. 7 A BCNF D-tree schema for a data base in Fig.2.

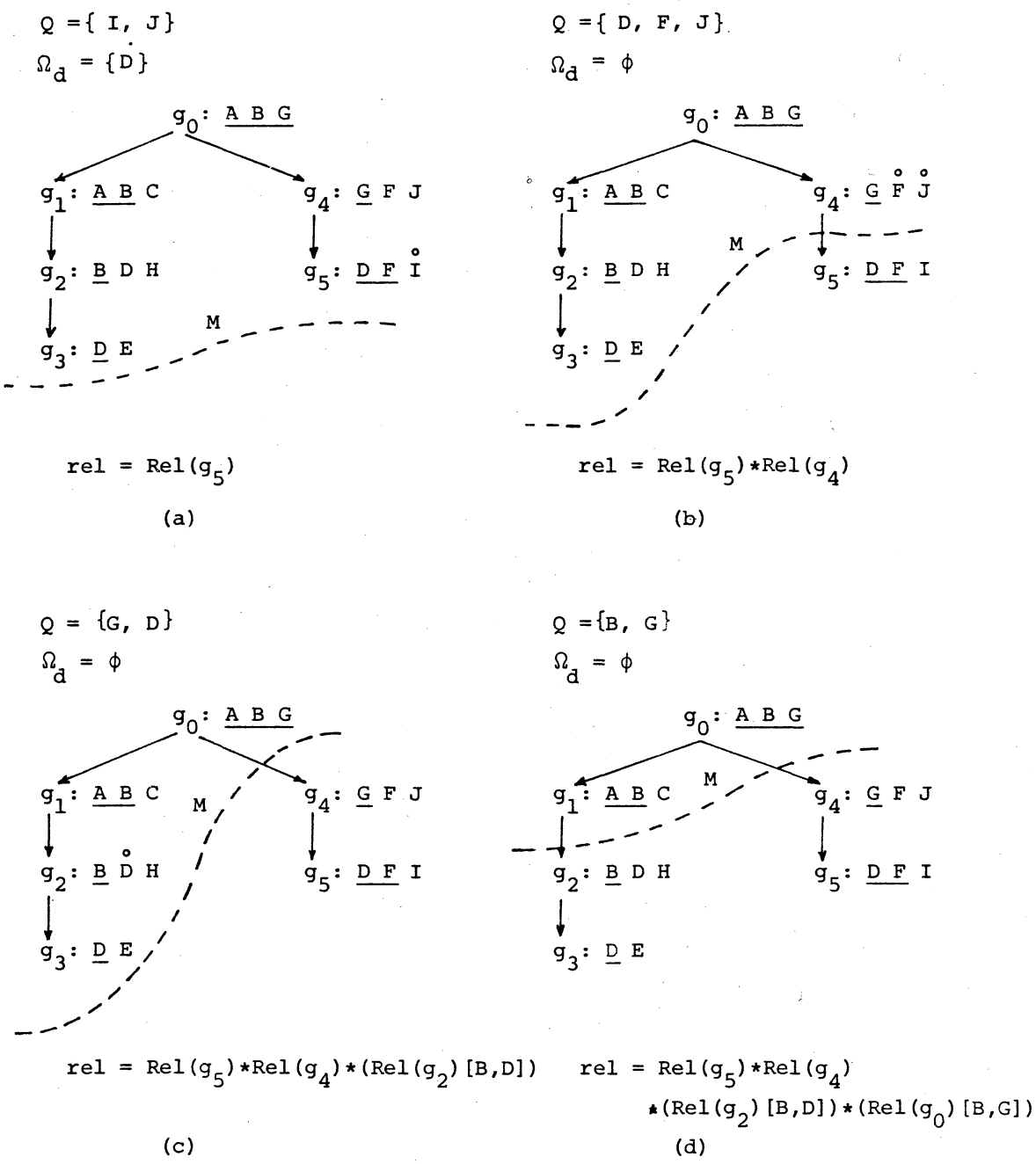


Fig. 8 The composition process of a relational expression for Q={I, J} in the example data base Fig.7.

Ω : {A, B, C, D, E}

F : AB $\rightarrow$ C

D $\rightarrow$ BE

M : B $\twoheadrightarrow$ AC | DE in Ω

(a) an example scheme (Ω, (F, M))

F' : AB $\rightarrow$ ABC

D $\rightarrow$ DBE

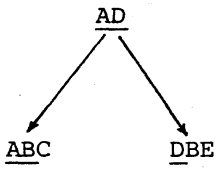
AD $\rightarrow$ ADBCE

M'' : B $\twoheadrightarrow$ AC | DE in Ω

AB $\twoheadrightarrow$ DE in Ω

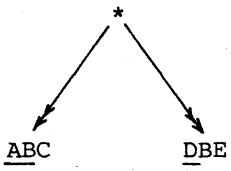
D $\twoheadrightarrow$ AC in Ω

(b) the closure $(F, M)^{\dagger}$



(c) decomposition with priority to FD's

AD is redundant.



(d) decomposition with priority to LMVD's.
(decomposition by
B $\twoheadrightarrow$ AC | DE in Ω)

Fig. 9 An example of a data base in which the decomposition with priority to FD's causes redundancy.

Ω : {A, B, C, D, E}

F : B $\rightarrow$ C

M : AB $\twoheadrightarrow$ CD | E in Ω

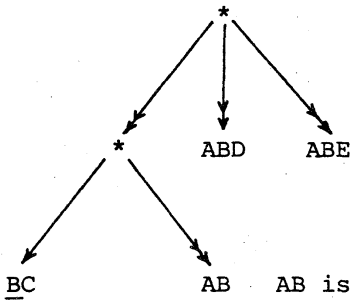
(a) an example scheme (Ω, (F, M))

F' : B $\rightarrow$ BC

M'' : AB $\twoheadrightarrow$ C | D E in Ω

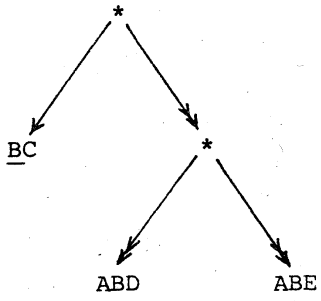
B $\twoheadrightarrow$ ADE in Ω

(b) the closure $(F, M)^{\dagger}$



AB is redundant.

(c) decomposition with priority to LMVD's.

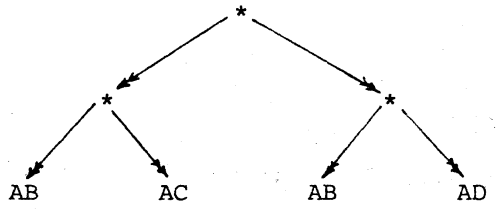


(d) decomposition with priority to FD's.

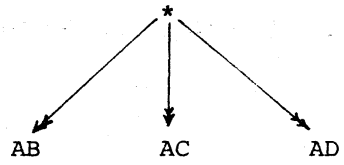
Fig. 10 An example of a data base in which the decomposition with priority to LMVD's causes redundancy.

M'' : AB ↠ C | D in ABCD
A ↠ B | C | D in ABCD

(a) an example M''



(b) AB ↠ C | D then A ↠ B | C | D
(AB is redundant)



(c) A ↠ B | C | D

Fig. 11 Decomposition order and redundancy.

Ω = {<Employee>, <Salary>, <Department>, <Item>, <Supplyer>, <Skill>}

F : <Employee> ---→ <Salary><Department>

M : <Department> -↠ <Employee><Salary><Skill> | < Item><Supplyer> in Ω
<Employee> -↠ <Skill> in Ω

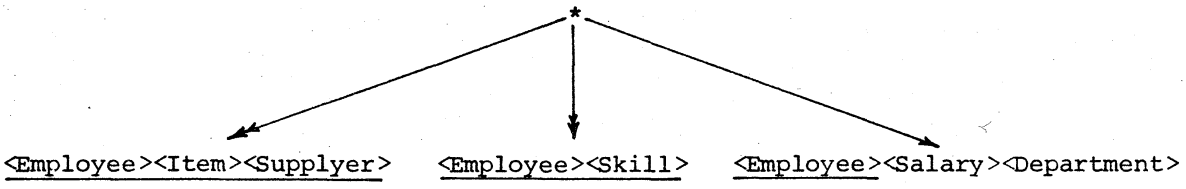
(a) an example scheme (Ω, (F, M))

F' : <Employee> ---→ <Salary><Department>

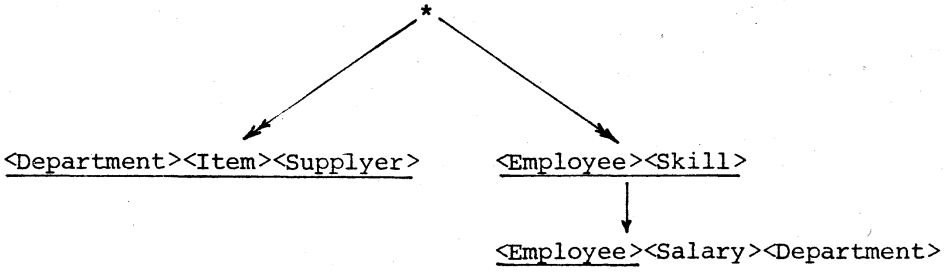
M'' : <Department> -↠ <Employee><Salary><Skill> | <Item><Supplyer> in Ω
<Employee> -↠ <Skill> | <Item><Supplyer> in Ω

(b) the closure (F, M)†

Fig. 12 An example of a data base and the closure of dependencies.



(a) decomposition with the application of
<Employee> ↠ <Skill> | <Item><Supplyer> at first



(b) decomposition with the application of
<Department> ↠ <Employee><Salary><Skill> | <Item><Supplyer>
at first

Fig. 13 Comparison of two decompositions of the data base in Fig.11.

Ω = {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}

F : G -→ DKLM
AC -→ OPQ
H -→ ABN

M : AB ↠ CDEFKLM | GHIJNOPQ
C ↠ AELM | BFOP
D ↠ AHL | BIJMOPN
F ↠ ABG | HIJLM
HC ↠ AD | BEF
K ↠ LM | QPAB
L ↠ QP | C
M ↠ ON | C

Fig. 14 An example scheme (Ω, (F, M)).

F' :
G --> BDGKLMNOP
AC --> ACLMOPQ
H --> ABDHKLMNOPQ
AB --> ABDKLMNOPQ
C --> ACLMOP
D --> DBMNOP
AD --> ADBKLMNOPQ
F --> ABDFKLMNOPQ
K --> KLMOP
L --> LP
M --> MO

M' :
G [BDKLMNOP] ACEFHIJQ
AC [LMOPQ] BDEFGHIJKN
H [ABDKLMNOPQ] CEFGIJ
AB [DKLMNOPQ] CEF | GHIJ
C [ALMOP] E | BF
D [BMNOP] AHL | IJ
F [ABDKLMNOPQ] G | HIJ
K [LMOP] ABCDEFGHIJNQ
L [P] ABCDEFGHIJKMNOQ
L [P] Q | C
M [O] ABCDEFGHIJKLNPQ
M [O] N | C
C [ALMOP] BDEFGHIJKNQ
D [BMNOP] ACEFGHIJKLQ
F [ABDKLMNOPQ] CEGHIJ
AD [BKLMNOPQ] CEFGHIJ

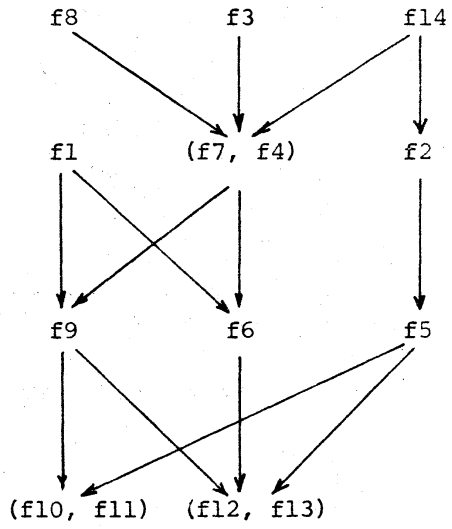
Fig. 15 Calculation of (F', M') from the scheme in Fig.14.

M'' :
G [BDKLMNOP] ACEFHQ | IJ
H [ABDKLMNOPQ] CEF | G | IJ
AB [DKLMNOPQ] CEF | G | H | IJ
F [ABDKLMOPQ] G | H | IJ | CE
C [ALMOP] E | BDFGHIJKLMNQ
D [BMNOP] IJ | ACEFGHKLQ
CB [ADKLMNOPQ] E | F | G | H | IJ
DA [BKLMNOPQ] IJ | CEF | G | H
L [P] Q | C
M [O] N | C

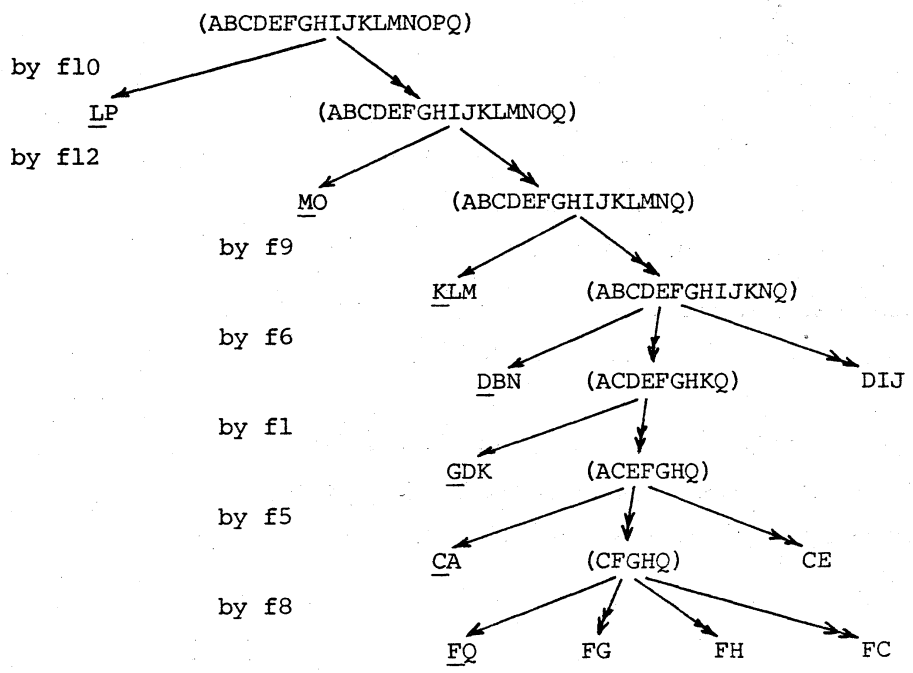
Fig. 16 Calculation of M'' for the scheme in Fig.14.

M^* : f1 : G [BDKLMNOP] ACEFHQ | IJ
f2 : AC [LMOPQ] BDEFGHIJKN
f3 : H [ABDKLMNOPQ] CEF | G | IJ
f4 : AB [DKLMNOPQ] CEF | G | H | IJ
f5 : C [ALMOP] E | BDFGHIJKLMNQ
f6 : D [BMNOP] IJ | ACEFGHKLQ
f7 : AD [BKLMNOPQ] IJ | CEF | G | H
f8 : F [ABDKLMNOPQ] G | H | IJ | CE
f9 : K [LMOP] ABCDEFGHIJNQ
f10 : L [P] ABCDEFGHIJKMNOQ
f11 : L [P] Q | C
f12 : M [O] ABCDEFGHIJKLNPQ
f13 : M [O] N | C
f14 : CB [ADKLMNOPQ] E | F | G | H | IJ

(a) M^*



(b) dependency diagram



(c) 4NF D-tree schema

Fig. 17 An example 4NF D-tree schema obtained from $(\Omega, (F, M))$ in Fig.14.

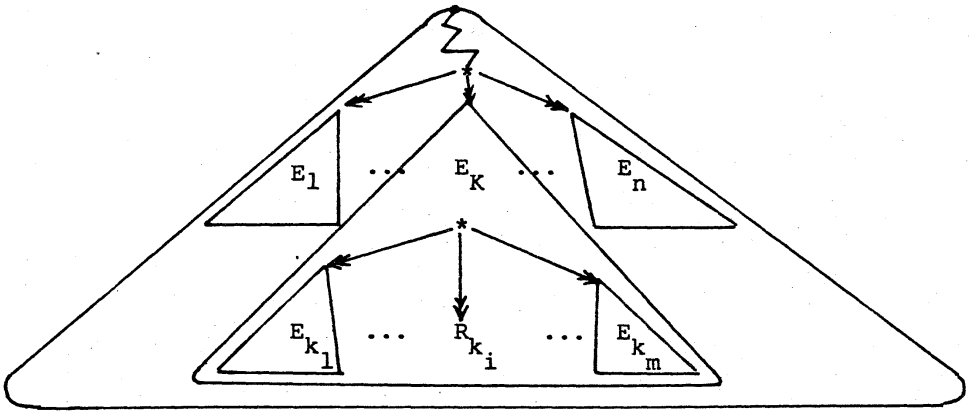
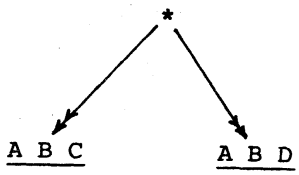


Fig. 18 A general form of a decomposition tree.

M : A B $\twoheadrightarrow$ C | D
B $\twoheadrightarrow$ C | D

4NF D-tree schema



Query : Q = {C, D}

automated processing without
any consideration for hidden LMVD's

(ABC * ABD)[C, D]

irredundant expression

(BC * BD)[C, D]

Fig. 19 The necessity of consideration for hidden LMVD's to compose an irredundant relational expression for a given query.