THE DESIGN OF ¬1NF RELATIONAL DATABASES INTO NESTED NORMAL FORM

Mark A. Roth and Henry F. Korth

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-86-27 December 1986

The Design of ¬1NF Relational Databases into Nested Normal Form

Mark A. Roth

Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

Henry F. Korth†

Department of Computer Science
University of Texas at Austin
Austin, TX 78712-1188

Abstract

We develop new algorithms for the design of non first normal form relational databases that are in nested normal form. Previously, a set of given multivalued dependencies and those multivalued dependencies implied by given functional dependencies were used to obtain a nested normal form decomposition of a scheme. This method ignored the semantic distinction between functional and multivalued dependencies and utilized only full multivalued dependencies in the design process. We propose new algorithms which take advantage of this distinction, and use embedded multivalued dependencies to enhance the decomposition. This results in further elimination of redundancy due to functional dependencies in nested normal form designs.

1. Introduction

Interest in generalizations of the relational data model has developed rapidly in recent years. Many researchers have observed that first normal form (1NF) relations are not appropriate for recent database applications such as information retrieval systems [Sch1, SP], computer aided design [BaKi], forms management [SLTC], and statistical databases [OO]. Utilizing relational databases for such applications requires non-first normal form (¬1NF) or ***nested*** relations, whose tuple components can be sets or even relations themselves. During the last few years there have been several important results regarding the loosening of the 1NF restriction on relational databases. Seminal work in this area can be found in [Mak, KTT, FV, OY] (extended dependencies and normal forms), [AB, FT, JS, RKS] (extended algebra and calculus query languages), and [Sch2, D+] (implementations).

The following example illustrates a simple ¬1NF relation.

Example 1.1: Consider the following ¬1NF database scheme.

$$\begin{aligned} \text{Emp} &= (\text{ename, Children, Skills}),\\ \text{Children} &= (\text{name, dob}),\\ \text{Skills} &= (\text{type, Exams}),\\ \text{Exams} &= (\text{year, city}). \end{aligned}$$

In this scheme each employee has a set of children each with a name and birthdate, and a set of skills, each with a skill type and a set of exam years and cities, when and where the employee retested his proficiency at

† Research partially supported by an IBM Faculty Development Award and NSF grant DCR-8507224

Employee

ename	Children		Skills		
	name	dob	type	Exams	
				year	city
Smith	Sam	2/10/84	typing	1984	Atlanta
	Sue	1/20/85		1985	Dallas
			dictation	1984	Atlanta
Watson	Sam	3/12/78	filing	1984	Atlanta
				1975	Austin
				1971	Austin
			typing	1962	Waco

Figure 1-1. A sample relation on the Emp scheme.

the skill. Using the terminology of nested relations, we say that an Employee tuple t_1 has 3 attributes. The Children attribute of t_1 is a nested relation whose scheme is Children = (name, dob). Likewise, the Skills attribute of t_1 is a nested relation whose scheme is Skills = (type, Exams). A tuple t_2 in a Skills nested relation has 2 attributes. The Exams attribute of t_2 is a nested relation. A sample relation is shown in the relation in Figure 1-1. □

The benefits of properly structured nested relations can be described in terms of the properties of the functional and multivalued dependencies that hold. For example, let U be a set of attributes, X, Y, and Z a disjoint partition of U, and r a 1NF relation on scheme $R = (U)$. If the multivalued dependency $X \rightarrow\!\!\!\!\rightarrow Y|Z$ holds in r then consider the relation s with the Z attributes forming one nested relation and the Y attributes forming another nested relation for each X value. Relation s is a $\neg$1NF relation with several good properties. First, X is a key for s, giving a unique tuple in s for each X-value. Second, the Y and Z nested relations are independently updatable; adding a value to Z (Y) automatically enforces the underlying MVD by matching all values in Y (Z) with the new value added.

A third benefit of a properly designed nested scheme relates to the properties of the extended relational algebra operator ***nest*** [JS]. The nest operator is a grouping operator that creates nested relations. If in our example, we start with relation r and wish to form relation s, we need to apply the nest operator twice, once to form the nested relation on attributes of Z and once to form the nested relation on attributes of Y. In general the order in which this nesting is performed matters. However, it has been shown [JS] that if the MVD $X \rightarrow\!\!\!\!\rightarrow Y|Z$ holds in r, then we obtain the same value for relation s regardless of which nest is performed first.

In this paper, we address the issue of the design of $\neg$1NF relational databases by studying an important normal form called ***nested normal form*** [OY]. Normal forms for $\neg$1NF relations must satisfy two requirements. As in the case of 1NF relations, normal forms must reduce redundancy. In addition a nested relation has a nesting structure which implies certain semantic connections among the attributes. Thus, "a normal form for nested relations aims not only to group attributes into related sets of attributes, but also to choose a nesting structure which is a good representation of the set of semantic connections that already exist in the real world among attributes." [OY; 2]

We present new algorithms for the design of $\neg$1NF databases. Our algorithms produce designs that are in nested normal form (NNF). It is frequently the case that several NNF designs are possible for a given

database scheme. Previous NNF design algorithms considered only a subset of the set of all possible NNF designs. This may result in the generation of an intuitively unappealing design despite the existence of a more attractive alternative. (We will show an example of such a situation.) Our algorithms allow us to find NNF designs that are both intuitively appealing and that result in less redundancy than those generated by previous techniques. The power of our techniques derives from that fact that our algorithms are able to take into account the semantic distinction between functional dependencies and multivalued dependencies, and allow for the use of embedded multivalued dependencies. The final two sections include proofs of the correctness of our algorithms along with modifications to our algorithms that offer additional elimination of redundancy.

2. Definitions

In this section we present some basic definitions regarding multivalued dependencies that we will use in subsequent sections. Further details can be found in [Mai].

Let r be any relation on scheme R, with X and Y subsets of R and $Z = R - XY$. Relation r ***satisfies*** the ***multivalued dependency*** $X \twoheadrightarrow Y$ if for every pair of tuples t_1 and t_2, in r, if $t_1[X] = t_2[X]$, then there exists a tuple t_3 in r with $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$, and $t_3[Z] = t_2[Z]$.

An MVD is said to be ***embedded*** if the MVD holds on a projection of the relation. Let r be any relation on scheme R, $Z \subseteq R$, and $X \subseteq Z, Y \subseteq Z$. Relation r satisfies the ***embedded multivalued dependency*** (EMVD) $X \twoheadrightarrow Y|Z - XY$ when the MVD $X \twoheadrightarrow Y$ holds in the projection of r onto Z. If an MVD or EMVD, G, holds in a relation r with attributes Z, then the projection of that dependency on a set of attributes $Y \subseteq Z$, denoted $proj_Y(G)$, holds in the projection of r onto Y if and only if the left hand side of G is a subset of Y. A dependency is projected on Y by eliminating all attributes on the right hand side that are not in Y.

Let U be the universe of attributes, X a set of attributes, and M a set of multivalued dependencies. A ***dependency basis*** for X, denoted $DEP_M(X)$, or $DEP(X)$ when M is understood, is a partition of $U - X$ into sets of attributes $Y_1, Y_2, \ldots, Y_n$, such that if $Z \subseteq U - X$, then $X \twoheadrightarrow Z$ if and only if Z is the union of some of the Y_i's.

For the MVD-set $\{X \twoheadrightarrow V_i \mid i = 1, \ldots, n\}$ we shall often write $X \twoheadrightarrow V_1|V_2|\ldots|V_n$. If the V_i form a dependency basis for X then $X \twoheadrightarrow V_1|V_2|\ldots|V_n$ is called ***full*** [BFMY] and a set of MVDs containing only full MVDs is called a ***full set of MVDs***. We use $LHS(M)$ to denote the set of left hand sides of the MVDs in a set M of MVDs.

For a set M of MVDs, M^+ denotes the ***closure*** of M, i.e., the set of all MVDs that are implied by M. Given two sets of M and N of MVDs, M is a ***cover*** of N if $M^+ = N^+$. Many times we want to work with a ***minimum cover*** for a set of MVDs.

Definition 2.1: [OY] Given a set M of MVDs over U, an MVD $X \twoheadrightarrow W$ in M^+ is said to be

(a) ***trivial*** if $XW = U$, $W = \emptyset$ or $W \subseteq X$,

(b) ***left-reducible*** if $\exists X', X' \subset X$, such that $X' \twoheadrightarrow W$ is in M^+,

(c) ***right-reducible*** if $\exists W', W' \subset W$, such that $X \twoheadrightarrow W'$ is in M^+,

(d) ***transferable*** if $\exists X', X' \subset X$, such that $X' \twoheadrightarrow W(X - X')$ is in M^+.

An MVD $X \twoheadrightarrow W$ is said to be ***reduced*** if it is nontrivial, left-reduced, right-reduced, and nontransferable. A set of MVDs M is said to be a ***minimum cover*** if every MVD in M is reduced, and no proper subset of

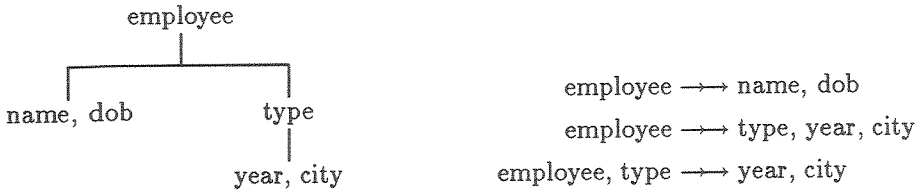


Figure 3-1. Scheme tree and implied MVDs for employee database.

M is a cover of M.

Let M^- be the set of all reduced MVDs implied by M, N be a set of MVDs, and M be a minimal cover of N. Then elements in $LHS(M^-)$ are called ***keys*** of N, and the elements in $LHS(M)$ are called ***essential keys*** of N. Elements in $LHS(M^-) - LHS(M)$ are called ***nonessential keys*** of N. Note that we are using the term key in a nonstandard sense, following [OY]. We shall find that it is desirable to design a ¬1NF scheme in which all attributes of a key (in the sense defined above) appear together in one relation scheme, rather than being ***split*** among several. This motivates the following definitions:

Definition 2.2: [BFMY] A MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ ***splits*** Z if both $Z \cap Y$ and $(U - X - Y) \cap Z$ are nonempty. A set M of MVDs is ***conflict-free*** if no MVD in M splits a key of M and for all pairs X and Y in $LHS(M)$, $(DEP(X) \cap DEP(Y)) \subseteq DEP(X \cap Y)$.

A relationship which states that certain projections of a relation must join (natural join) to the original relation is called a ***join dependency*** (JD). Let r be any relation on scheme R and let $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ be a set of schemes which are projections of scheme R. Relation r satisfies the ***join dependency*** $\bowtie (R_1, R_2, \ldots, R_n)$ if r decomposes losslessly onto $R_1, R_2, \ldots, R_n$. That is,

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_n}(r).$$

If a join dependency is equivalent to a set of multivalued dependencies then the $\mathcal{R}$ is an ***acyclic*** set of schemes. Acyclic schemes have several good properties described in [BFMY, Sac], including the fact that the set of multivalued dependencies which are equivalent to a join dependency are conflict free. Properties of conflict free dependencies are described in [AC, BeK1, BeK2, Sci1, Sci2, Sci3]. Sciore [Sci2] states that in "real world" situations, every "natural" set of MVDs must be conflict free. Conflict free sets have the desirable property that they allow a unique fourth normal form dependency preserving database scheme; moreover, non-conflict free sets have no such normalization.

3. Nested Normal Form

Özsoyoğlu and Yuan [OY] introduced the first comprehensive approach to normalization for ¬1NF relations. They consider nested relations whose schemes are structured as trees, called ***scheme trees***, and introduce a normal form for such relations, called ***nested normal form*** (NNF). A ***scheme tree*** is a tree whose vertices are labeled by pairwise disjoint sets of non-nested attributes, where the edges of the tree represent MVDs between the attributes in the vertices of the tree. These MVDs allow a 1NF relation to be represented as a ¬1NF relation with the desirable properties discussed in the previous section. The scheme tree and associated MVDs for the Emp scheme are shown in Figure 3-1.

We now introduce the notation we shall use to discuss scheme trees formally. Let U be a set of non-nested attributes, let T be a scheme tree, and let $S(T)$ denote the set of all attributes in T. $S(T)$ is a subset of U. Each edge of scheme tree T represents a multivalued dependency on $S(T)$. Let $e = (u, v)$ be an edge in T. The MVD on $S(T)$ represented by the edge e is $A(u) \twoheadrightarrow D(v)$, where $A(u)$ is the union of all ancestors of u (including u), and $D(v)$ is the union of all descendants of v (including v). Since it may be the case that $S(T)$ is a proper subset of U, the MVD represented by an edge of T may be an embedded MVD on U. We denote by $MVD(T)$ the set of MVD's represented by the edges of T.

Definition 3.1: [OY] Let T be a scheme tree, and $u_1, u_2, \ldots, u_n$ be all the leaf nodes of T. Then the ***path set*** of T, denoted $P(T)$, is $\{A(u_1), A(u_2), \ldots, A(u_n)\}$.
Note that, for a leaf node u, $A(u)$ is the union of all the nodes in the path from the root of T to u in T.

The following proposition gives some properties of a scheme tree.

Proposition 3.1: [OY] If T is a scheme tree, then
1. $P(T)$ is an acyclic database scheme,
2. The set $MVD(T)$ of multivalued dependencies is equivalent to the join dependency $\bowtie(P(T))$, and
3. $MVD(T)$ is a conflict free set of MVDs. □

Let T be a scheme tree with respect to M, where $S(T) \subseteq U$, and (u, v) be an edge in T. Assume there is a key X of M such that there exists $Z \in DEP(X)$ and $D(v) = Z \cap S(T)$. Then, v is said to be a ***partial redundant*** in T with respect to X if $X \subset A(u)$. The MVD, $X \twoheadrightarrow D(v)$ in the context of $S(T)$ is a ***partial dependency*** in $S(T)$. Similarly, if there exists some sibling nodes $v_1, v_2, \ldots, v_n$ of v in T such that $W = \bigcup_{i=1}^{n} D(v_i)$, $X \subset A(u) \cup W$, and M does not imply $XW \twoheadrightarrow D(v)$ in the context of $S(T)$, then v is said to be ***transitive redundant*** with respect to X in T. In this case, the MVD, $X \twoheadrightarrow D(v)$, in the context of $S(T)$, is said to be a ***transitive dependency*** in $S(T)$.

Partial and transitive dependencies must be removed from scheme trees during the design process. In our final design, we want each edge (u, v) to represent $A(u) \twoheadrightarrow D(v)$. If a partial dependency holds, it has a LHS that is a proper subset of $A(u)$ for some u, and thus, edge (u, v) does not represent that (partial) dependency. Indeed, the partial dependency implies the dependency represented by (u, v). Partial dependencies can be eliminated by splitting a tree into two or more trees. If a transitive dependency holds in T, then there is a MVD that holds on the attributes of T that does not follow from the MVDs represented by edges of some subtree of T.

In order to avoid dividing keys in trees, we will use the notion of a fundamental key. Let M be a set of MVDs on U and $V \subseteq U$. The set of ***fundamental keys*** on V, denoted $FK(V)$, is defined by:

$$FK(V) = \{V \cap X | X \in LHS(M) \text{ and } V \cap X \neq \emptyset, \text{ and there is no } Y \in LHS(M) \text{ such that } V \cap X \supset V \cap Y \text{ and } V \cap Y \neq \emptyset\}.$$

Intuitively, the set of fundamental keys on V consists of those keys that have minimal, nonempty intersections with V.

It is desirable to design scheme trees in which each node is a fundamental key for its subtree, since it has been shown [OY] that the path set of such a tree is a fourth normal form design. Given a set M of MVDs on attributes U, [OY] gives an algorithm to decompose U into a set of ***normal*** scheme trees which do not have partial or transitive redundancies and do not split keys among nodes in the trees. Formally, a normal scheme tree is defined as follows.

Definition 3.2: A scheme tree T is said to be ***normal*** with respect to a set of MVDs, M, if

1. M implies $MVD(T)$,
2. There are no partial dependencies in T.
3. There are no transitive dependencies in T.
4. The root of T is a key, and for each other node u in T, if $FK(D(u)) \neq \emptyset$, then $u \in FK(D(u))$.

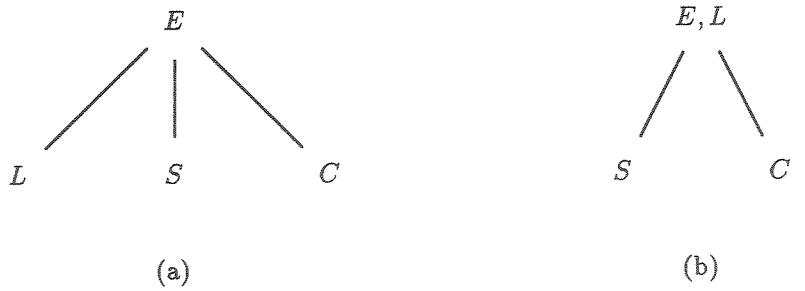
[OY] uses a decomposition technique to construct a NNF design. Decomposition is performed when a path set is not in 4NF by choosing a fundamental key K on a node V and splitting V into a subtree with root K and one child of K for each element of the dependency basis of K with respect to V. This decomposition procedure is repeated until no further non-4NF path set exists. This method uses MVDs and the MVD counterpart of FDs as input to the NNF decomposition algorithm. In [YO], the authors have combined FDs and MVDs into an ***envelope*** set of dependencies. They propose that this envelope set could be used as input to a slightly modified NNF algorithm which would then take into account the different semantics of FDs and MVDs. Using the algorithm in [OY], singleton sets are likely to appear when FDs are used to perform the decomposition. We propose a new method for achieving nested normal form which takes into account the different semantics of FDs.

4. Examples of the Nested Normal Form Design Problem

In this section we provide a series of examples to illustrate the problems with prior NNF design techniques and to show intuitively how our algorithms produce better designs.

Our first example shows that if we ignore FDs and use only their MVD counterparts (i.e., $X \longrightarrow\!\!\!\!\!\rightarrow Y$ instead of $X \rightarrow Y$), then we may obtain nested relations whose schemes are singleton sets.

Example 4.1: Let $U = ELSC$ and $D = \{E \longrightarrow\!\!\!\!\!\rightarrow S, E \rightarrow L\}$, where E is an employee id, L is the employee's location, S is an employee's skill, and C is an employee's child. Using the method of [OY], we would use the MVD $E \longrightarrow\!\!\!\!\!\rightarrow L$ implied by $E \rightarrow L$ and create the $\neg$1NF relation with scheme tree shown in Figure 4-1a. However, since $E \rightarrow L$, each L-set created by this scheme will be a singleton set. Therefore, we should use the scheme tree of Figure 4-1b. □



(a) (b)

Figure 4-1. Scheme trees for Example 4.1 using approach (a) of [OY], and (b) modified for different FD semantics.

Although an approach to better handling FDs in an NNF design is being pursued [Ozs], the introduction of embedded MVDs has not yet been considered. One reason for this is that the implication problem for EMVDs has not been solved. That is, given a finite set of attributes U, there is no known complete

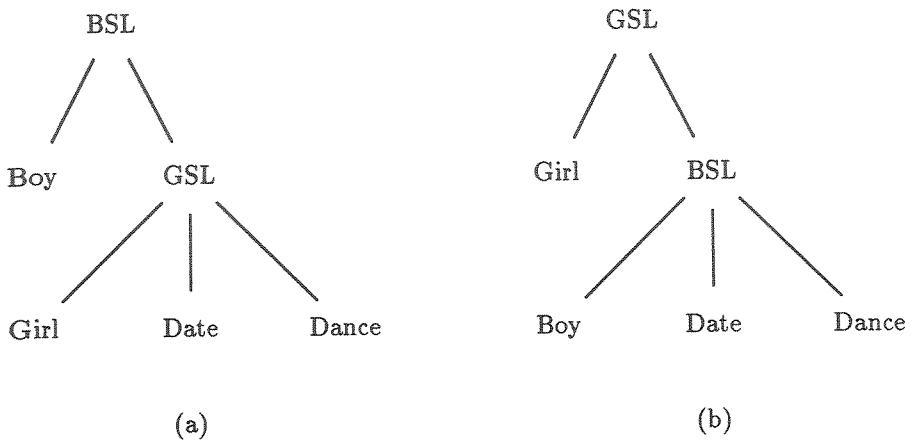


Figure 4-2. Two initial scheme trees for Example 4.2,
using (a) BSL, and (b) GSL to decompose.

axiomatization of EMVDs. Furthermore, if the set of attributes is infinite, then there is provably no complete axiomatization [PP]. There are, however, several sound inference rules for EMVDs and for EMVDs together with MVDs and FDs. Thus, if we are given that an EMVD should hold in our database, we can use that knowledge, plus any dependencies derivable from the known inference rules, to improve our database design. One of the contributions of our new approach to NNF design is to include EMVDs in the set of input dependencies.

In addition to including EMVDs in the design, we take a different approach to the design of NNF relations, which gives the designer of a ¬1NF scheme more control over the final outcome. As proved in [OY], the design scheme produces nested relations with a unique 4NF path set if and only if the input dependencies are conflict free. Otherwise, there are several designs which will satisfy the NNF requirements, not all of which will have 4NF path sets.

In the approach of [OY], different designs are generated depending on the fundamental keys selected to decompose a set of attributes into several branches of the scheme tree, and depending on the chosen ordering of all keys used in testing for partial and transitive dependencies and essential dependents. These different orderings of keys may cause different scheme trees to be split apart.

Example 4.2: Consider a scouting database with attributes BSL (boy scout leader), GSL (girl scout leader), Boy, Girl, Date (when a Boy and Girl went out to eat), and Dance (when a Boy and Girl went to a dance). The dependencies which are assumed to hold in this database are BSL $\twoheadrightarrow$ Boy, GSL $\twoheadrightarrow$ Girl, (Boy, Girl) $\twoheadrightarrow$ Date, and (Boy, Girl) $\twoheadrightarrow$ Dance. We also have the EMVD $\emptyset \twoheadrightarrow$ BSL | GSL, however EMVDs are not considered in the approach of [OY]. Following the approach of [OY] we create an initial scheme tree by decomposing the entire set of attributes based on the fundamental keys { BSL, GSL, (Boy, Girl)}. The decomposition algorithm arbitrarily chooses either of these keys to perform the initial decomposition, and depending on which one is chosen quite different NNF designs result. The two initial decompositions which result from using BSL or GSL are shown in Figure 4-2.

When BSL is used to decompose the initial scheme tree (Figure 4-2a), the tree has a partial dependency GSL $\twoheadrightarrow$ Girl, and so the edge (GSL, Girl) is removed from Figure 4-2a, and a new tree created with the

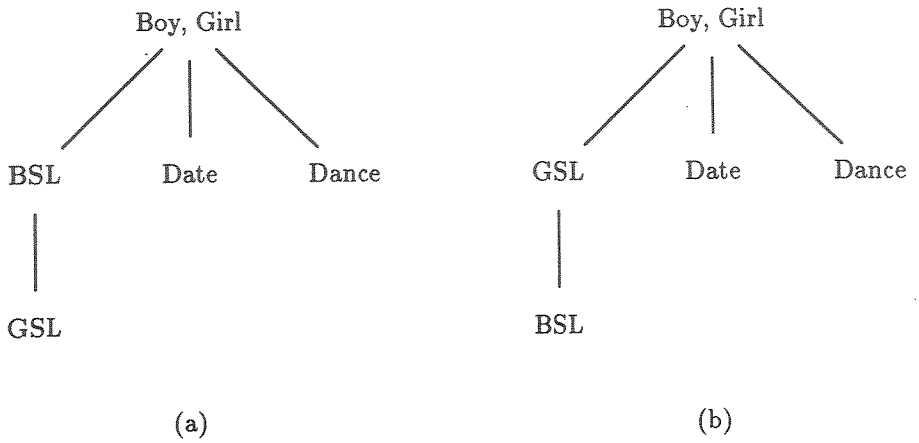


Figure 4-3. Two alternative trees using (Boy, Girl) to start decomposition, and using (a) BSL, and (b) GSL to further decompose.

single edge (GSL, Girl). Similarly, when GSL is used to decompose the initial scheme tree (Figure 4-2b), the tree has partial dependency BSL $\rightarrow\!\!\!\!\rightarrow$ Boy, and so the edge (BSL, Boy) is removed from Figure 4-2b, and a new tree created with the single edge (BSL, Boy). The scheme trees which result in these two cases have 4NF path sets (BSL, Boy), (BSL, GSL, Date), (BSL, GSL, Dance), and (GSL, Girl). Although the path sets form a 4NF design, in neither case is the particular decomposition very intuitive. Take the trees which result from starting with BSL. The scheme trees show that for each boy scout leader there is a set of boys and a set of girl scout leaders. And for each girl scout leader associated with a boy scout leader there is a set of Dates and a set of Dances. Also for each girl scout leader there is a set of girls. The relationship between BSL, GSL and Date and Dance is only an indirect one via the leaders associated boys and girls. Nevertheless, these two schemes are the ones recommended by [OY].

Let us consider the alternative of using (Boy, Girl) as the fundamental key to start the decomposition. This choice is not allowed by [OY] since the MVDs with left hand sides (Boy, Girl) are split by the other given MVDs, and this will result in a path set which is not 4NF. However, let us explore what happens if we do use this fundamental key, primarily to compare later with results achieved by our design approach for this problem. Using (Boy, Girl) as the fundamental key two alternatives for the initial decomposition are shown in Figure 4-3. The two alternatives result from a decision to use either BSL or GSL as the fundamental key when decomposing node (BSL, GSL).

When BSL is used to decompose (BSL, GSL) (Figure 4-3a), we find the partial dependency (BSL, Girl) $\rightarrow\!\!\!\!\rightarrow$ GSL in the scheme tree, and so we remove edge (BSL, GSL) and create a new scheme tree with edge ((BSL, Girl), GSL). Similarly, if GSL is used to decompose (BSL, GSL), (Figure 4-3b) we find the partial dependency (GSL, Boy) $\rightarrow\!\!\!\!\rightarrow$ BSL in the scheme tree, and so we remove edge (GSL, BSL) and create a new scheme tree with edge ((GSL, Boy), BSL). In both cases the path set is not in 4NF. In the first case, (Boy, Girl, BSL) is decomposable by MVD BSL $\rightarrow\!\!\!\!\rightarrow$ Boy, and in the second case, (Boy, Girl, GSL) is decomposable by MVD GSL $\rightarrow\!\!\!\!\rightarrow$ Girl. However, the resulting scheme trees are in nested normal form. In these cases, the initial decomposition seems better in that we have the Date and Dance attributes directly associated with the (Boy, Girl) pair. However, the additional tree that is created to solve the partial dependency presents

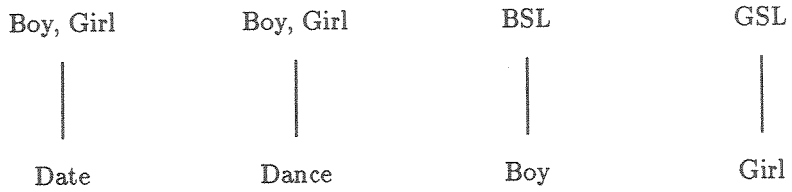


Figure 4-4. Initial scheme trees for each 4NF scheme of Example 4.3.

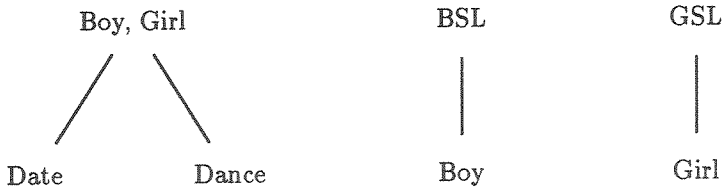


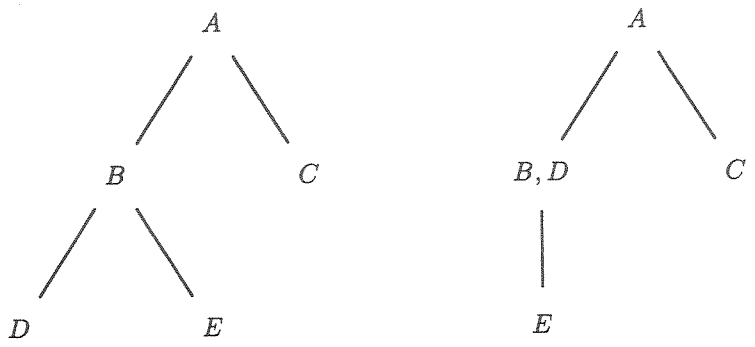
Figure 4-5. Final scheme trees using new approach to design of Example 4.3.

quite an unintuitive grouping of attributes. □

Our design algorithm will start with a 4NF decomposition and will preserve that decomposition throughout the remainder of the design. Thus, the primary point where different NNF designs will originate is embodied in the well studied and understood creation of a 4NF decomposition. We note, that when the input set of dependencies is conflict free there is a unique 4NF decomposition, and, therefore, our approach also produces an NNF design with a unique path set for this case. Before we present our approach formally, we show how it can be applied to the database of Example 4.2.

Example 4.3: (Continuation of Example 4.2.) We saw that it seemed best to ensure that Date and Dance were associated with the key (Boy, Girl) and so in the 4NF decomposition we use the key (Boy, Girl) to make the first split. Thus, we decompose into schemes (Boy, Girl, Date), (Boy, Girl, Dance), and (Boy, Girl, BSL, GSL). Now we can use the other two MVDs in any order to decompose (Boy, Girl, BSL, GSL) into (BSL, Boy), (GSL, Girl), and (BSL, GSL). Considering just the MVDs, this decomposition is in 4NF. If we consider the EMVD, as we propose to do, then the scheme (BSL, GSL) is decomposed into BSL and GSL, and these two schemes are eliminated since they are proper subsets of other schemes. Our method will then proceed to create a scheme tree for each scheme in the 4NF decomposition, as shown in Figure 4-4.

Then we combine scheme trees when the common attributes of two trees form the same root to non-leaf path in both trees. In this example, we combine the two trees with root (Boy, Girl) and our final design is a set of three scheme trees as shown in Figure 4-5. This design more clearly depicts the intended relationships and came about partially due to the fact that we carefully selected the 4NF decomposition that was appropriate for this case. In the approach of [OY], this kind of decision making can only go into the choice of fundamental key selection, and there is no way to produce the scheme trees of Figure 4-5, no matter what choices are made. We note that if we did not allow the EMVD to influence our 4NF decomposition, then we would have had an additional edge relating BSL and GSL in either the BSL–Boy tree or the GSL–Girl tree. These trees would still be more intuitive, and are equally unattainable using [OY]. □



(a) (b)

Figure 5-1. Scheme tree (a) T_1, and (b) T_2.

5. Fundamentals of Our NNF Design Algorithm

The first procedure we will need for our algorithm is a 4NF decomposition procedure. Several have been proposed, however we require one that deals with both FDs and MVDs and does not treat FDs as MVDs in the design, thereby ignoring the different semantics that FDs impose. Two approaches are available, one by Beeri and Kifer [BeK1] and Katsuno [Kat], and the other by Yuan and Özsoyoğlu [YO]. In the first approach, given a set D of FDs and MVDs, a new set M' of MVDs is formed by first obtaining the full version of the MVDs in D, and then replacing the left-hand side X of each MVD in the full version by the closure of X with respect to D.

In the second approach, given a set D of FDs and MVDs over a set U of attributes, a new set $E(D)$ of MVDs, called an ***envelope set***, is created, so that $E(D)$ represents the structural dependencies in D relevant to the design process.

Definition 5.1: The ***envelope set*** $E(D)$ of a set D of FDs and MVDs is

$$E(D) = \{X \twoheadrightarrow W | X \in LHS(D) \text{ and } W \in DEP_D(X) \text{ and } D \not\Rightarrow X \rightarrow W\}.$$

If a database scheme is 4NF with respect to $E(D)$ then it is also 4NF (BCNF if D has FDs only) with respect to D. Thus, a database scheme for D can be obtained by using $E(D)$ as input to any 4NF decomposition algorithm [Fag, Lie, GR, LT], without considering the FDs and MVDs separately.

We must consider which of these two approaches to the design of flat databases will help us most in forming better $\neg$1NF designs. As shown in Example 4.1, in an NNF design FDs cause singleton sets to appear if the MVD represented by an edge in a scheme tree is also an FD. In general, nesting is not necessary when for some edge (u, v), $u \rightarrow v$ holds. Consider the scheme tree T_1 shown in Figure 5-1a. If $B \rightarrow D$ holds, then each B value will have a single D value associated with it. Therefore, there is no need to nest D values allowing the tree T_2, shown in Figure 5-1b, which has a smaller structure and is consistent with T_1 in that $MVD(T_1) \Rightarrow MVD(T_2)$.

In the first approach described above, a similar operation takes place in the closing of the left hand sides of the MVDs. Attributes in the dependency basis of a left hand side X, which are functionally determined by X are moved to the left to form the closure of X. Looking at T_1 and T_2 we see that

$MVD(T_1) = \{A \twoheadrightarrow BDE|C, AB \twoheadrightarrow D, AB \twoheadrightarrow E\}$. If we make the MVDs full and close the left hand sides according to the FD $B \rightarrow D$, then we get the set $M' = \{A \twoheadrightarrow BDE|C, ABD \twoheadrightarrow E|C\}$. Clearly, these MVDs are the MVDs found in $MVD(T_2)$. Thus, the closure has the effect of associating functionally determined attributes with keys used to create the ¬1NF hierarchies.

In the second approach, the envelope set of MVDs is used to represent the FDs and MVDs. Here, components of full MVDs are eliminated if those components are also FDs. There is no attempt to associate functionally determined attributes with the keys and so the envelope set does not help in eliminating singleton sets from our designs. For the above example,

$$E(MVD(T_1) \cup \{B \rightarrow D\}) = \{A \twoheadrightarrow BDE|C, AB \twoheadrightarrow D|E|C, B \twoheadrightarrow ACE\},$$

and this set of MVDs would not help us in achieving T_2. Therefore, we adopt the first approach and use the set of MVDs obtained by closing the left hand sides of the MVDs implied by the given set of dependencies to obtain our initial decomposition. We use M' to represent the set of MVDs produced by this approach.

Since we desire to include EMVDs in our design algorithm, we must perform some additional steps to achieve our final decomposition. Let U be the set of attributes to be used in the design, D a set of given FDs and MVDs, and F a set of given EMVDs. First, using the known inference rules, we generate all FDs, MVDs, and EMVDs that are implied by the EMVDs or the EMVDs together with any known FDs or MVDs. The new FDs and MVDs are added to the original set D and the new EMVDs are added to the original set F. We compute M', the set of MVDs obtained by making the MVDs implied by D full and closing the left hand sides. We also close the left hand sides of the EMVDs in F using the FDs in D. We then use M' as input to one of the usual 4NF decomposition algorithms. This results in a set of schemes $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$.

We now have a set of schemes $\mathcal{R}$ which is in 4NF with respect to the MVDs M'. However, each scheme in $\mathcal{R}$ may have one or more FDs implied by D embedded within it. In their flat database design, [BeK1] use these FDs to synthesize a set of schemes for each scheme of $\mathcal{R}$, further eliminating redundancy by achieving a 3NF decomposition with respect to the MVDs M' ***and*** FDs in D. We do not want to take the additional step of synthesizing 3NF schemes when designing ¬1NF relations since organizing these schemes in a scheme tree will only introduce singleton sets. However, when we allow EMVDs to influence our design, we will have to consider further decomposition based on the FDs in each scheme of $\mathcal{R}$. The reason for this is that we can not use an EMVD in the design process unless at some stage in the decomposition it becomes a full MVD. For example, if $U = ABCDE$, then we can not use the EMVD $A \twoheadrightarrow B|CD$ until U is decomposed into a scheme which does not have E in it. Thus, we perform two checks for the EMVDs in F following our decomposition into $\mathcal{R}$ with respect to M'. First, if an EMVD, F_j, becomes a nontrivial MVD when F_j is projected onto some scheme in $\mathcal{R}$, say R_i, then F_j is used to decompose R_i, and we replace R_i with its decomposition in $\mathcal{R}$. This process is repeated until no more EMVDs can be used to decompose schemes in $\mathcal{R}$. Second, for each scheme R_i in $\mathcal{R}$ we perform a temporary 3NF synthesis on R_i obtaining the schemes $\mathcal{S} = S_1, S_2, \ldots, S_k$. If an EMVD, F_j, becomes a nontrivial MVD when F_j is projected onto some scheme in $\mathcal{S}$, say S_j, then we use F_j to decompose S_j and add the decomposition to $\mathcal{R}$. This continues until all schemes in $\mathcal{S}$ have been considered. If any schemes remain in $\mathcal{S}$ then we take the union of those schemes and replace R_i with this union. If all schemes still remained in $\mathcal{S}$ then we replaced R_i with R_i and no change was made to $\mathcal{R}$. The remainder of the ¬1NF design uses M', F, and $\mathcal{R}$.

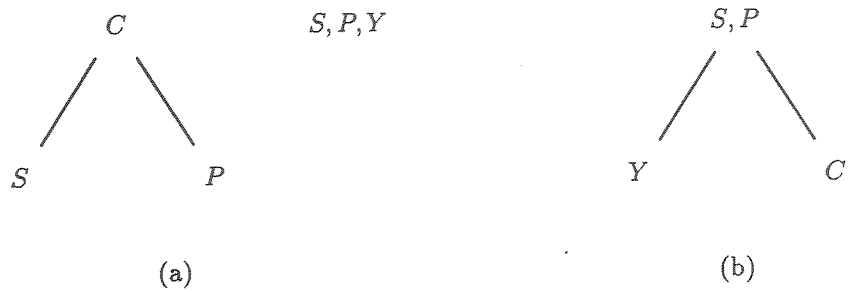


Figure 5-2. Scheme trees for Example 5.1 using (a) our approach, and (b) [OY].

Example 5.1: [Ull] Let $U = SPYC$, $D = \{SP \rightarrow Y\}$, and $F = \{C \twoheadrightarrow S|P\}$, where C is a course taken by a student S, and the course has prerequisite P taken by the student in year Y. There are no nontrivial FDs or MVDs implied by the EMVD, so we find $M' = \{SPY \twoheadrightarrow C\}$. Using this set as input to a 4NF decomposition algorithm, we get the scheme $SPYC$. Since this is the orginal set U, the EMVD is still an EMVD for this scheme. In the next step, we perform a synthesis on this scheme and get the decomposition $\{SPY, SPC\}$. Since, the EMVD in F is an MVD for scheme SPC, we use it to decompose SPC into CS and CP. The final decomposition is $\{CS, CP, SPY\}$, and using our NNF algorithm we get the scheme trees shown in Figure 5-2a. In comparison, the scheme tree produced by the NNF algorithm of [OY] is shown in Figure 5-2b. □

Once we have a decomposition $\mathcal{R}$ with respect to U and $M' \cup F$, we start the process of designing $\neg$1NF relations which are in nested normal form. We start by forming the trivial NNF design consisting of a single scheme tree for each 4NF scheme in $\mathcal{R}$. Each scheme tree is trivial since it will consist of a single path and, therefore, its edges will specify trivial EMVDs. In a later step, we will combine scheme trees to to achieve a nontrivial design.

In order to maintain NNF, even in a trivial design, we can not decompose each 4NF scheme arbitrarily. This is due to the requirement that only fundamental keys be used as the non-leaf nodes of a scheme tree. Thus, we use a simplified version of the DECOMP procedure in [OY] to perform the decomposition. The procedure is much simpler since the input is a set of attributes forming a 4NF scheme and there is no possibility of a split key appearing among these attributes (a condition checked for in the original procedure), and there is exactly one dependent in the dependency basis of any attribute set which is a subset of a 4NF scheme. We first provide a new definition for "fundamental keys," since we also need to deal with the set F of EMVDs which hold in the database. We also improve the definition by preferring fundamental keys which are not projections of some essential key. For example, if A and BC are essential keys and if we are finding the fundamental keys of ABD, then we would prefer to decompose based on the fundamental key A rather than B, since A represents a more complete relationship than B which is a projection of BC.

Definition 5.2: Given a set M' of MVDs, a set F of EMVDs, and a set U of attributes with $V \subseteq U$, the set of ***candidate fundamental keys*** of V, denoted $CFK(V)$, is defined as follows:

$$CFK(V) = \{W | W \in LHS(M') \vee (W \in LHS(\{F'\}) \wedge F' \in F \wedge proj_V(F') \text{ is a nontrivial MVD for } V)\}.$$

Out of $CFK(V)$ we prefer those keys that are minimal subsets of V and if there are none, we use the

minimal intersections of those keys with V. The ***preferred fundamental keys*** of V, denoted $PFK(V)$, and all ***fundamental keys*** of V, denoted $FK(V)$, are defined as follows:

$$PFK(V) = \{X | X \in CFK(V) \wedge X \subseteq V \wedge \nexists Y \text{ such that } Y \in CFK(V) \wedge Y \subset X\}$$

$$FK(V) = \{W | X \in CFK(V) \wedge W = X \cap V \wedge W \neq \emptyset \wedge \nexists Y \text{ such that } Y \in CFK(V) \wedge Y \cap V \subset W\}.$$

These modifications to the definition of fundamental keys allow for the fact that an EMVD could have been used to form a scheme with attributes V. Procedure DECOMP can now be specified as follows:

Procedure DECOMP(V, T)
    {V is a set of attributes which is a node in scheme tree T}
begin
    If V has 2 or more elements and $FK(V) \neq \emptyset$ then
        begin
          (1) If $PFK(V) \neq \emptyset$ then let $V_0 \in PFK(V)$ else let $V_0 \in FK(V)$;
          (2) $W = V - V_0$;
          (3) Change V into V_0 in T;
          (4) Attach W as a son of V_0;
          (5) DECOMP(W, T);
        end
end.

The final procedure we need for our design algorithm is a method for combining scheme trees while maintaining NNF and the original 4NF decomposition. We can combine scheme trees if the attributes that are in common to both trees form the same path, u to v, in each tree, where u is the root and v is a non-leaf node in both trees. If we have two trees that meet this requirement then we can temporarily merge them into a single tree. If the merged tree is free of transitive dependencies, then we let the merge become permanent. After making all possible merges, we have our final NNF design. The MERGE procedure is as follows:

Procedure MERGE(T_1, T_2, T_3)
    {T_1, T_2 are the two scheme trees whose common nodes form the same
     root to non-leaf path in both trees. T_3 is the merged tree.}
begin
    $T_3 := T_1$;
    For each edge (v, w) in T_2 do
        if (v, w) is not in T_3 then add (v, w) and (if necessary) nodes v and w to T_3
    end
end.

6. The NNF Design Algorithm

Using the procedures developed in the previous section, we can now specify our NNF design algorithm as follows:

Algorithm 1
    { input: a set of attributes U, a set of MVDs and FDs D, and a set of EMVDs F.
     output: a set of scheme trees $T_1, T_2, \ldots, T_n$ in NNF. }
begin
    (1) Find a 4NF decomposition of U with respect to $D \cup F$
        (a) Add to D any FDs and MVDs which can be inferred from $D \cup F$ using the EMVDs in F.
        (b) Add to F any EMVDs which can be inferred from $D \cup F$ using the EMVDs in F.
        (c) Find a 4NF decomposition $\mathcal{R} = (R_1, R_2, \ldots, R_k)$ with respect to M'.
        (d) Decompose schemes in $\mathcal{R}$ according to any EMVDs in F

which project as nontrivial MVDs on some scheme in $\mathcal{R}$.
Replace the decomposed schemes in $\mathcal{R}$ with their decomposition.

(e) For i:= 1 to k do
begin
(i) Synthesize a 3NF decomposition of R_i, $\mathcal{S} = (S_1, S_2, \ldots, S_m)$.
(ii) Decompose schemes in $\mathcal{S}$ according to any EMVDs in F
which project as nontrivial MVDs on some scheme in $\mathcal{S}$.
Remove any decomposed scheme from $\mathcal{S}$ and add the decomposition to $\mathcal{R}$.
(iii) Replace R_i in $\mathcal{R}$ with the union of the remaining schemes in $\mathcal{S}$.
end

(2) Prepare initial scheme trees.
(a) Initialize k scheme trees $T_1, T_2, \ldots, T_k$ with no edges and
single nodes labeled $R_1, R_2, \ldots, R_k$, respectively.
(b) For $i := 1$ to k do DECOMP(R_i, T_i) end.
(c) Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$.

(3) Merge trees.
Until no more changes can be made to $\mathcal{T}$ do
begin
(a) Select $T^1 \in \mathcal{T}$ and $T^2 \in \mathcal{T}$, $T^1 \neq T^2$,
such that T^1 and T^2 have not been considered together.
(b) If the common attributes of T^1 and T^2 from the same root to
non-leaf path in both trees then
begin
(i) MERGE(T^1, T^2, T^3).
(ii) If there are no transitive dependencies in T^3 then
$\mathcal{T} := \mathcal{T} - \{T^1, T^2\} \cup \{T^3\}$
end
end

end.

7. Correctness of Algorithm 1

In this section we show that the ¬1NF design produced by Algorithm 1 is in nested normal form. To do this we need to show that the four requirements of NNF hold for each relation in the design. Each scheme tree T of the design must satisfy the following four properties:

(1) Inference property: $D \cup F \Rightarrow MVD(T)$, where D is the input set of MVDs and F is the input set of EMVDs.
(2) PD property: There are no partial dependencies in T.
(3) TD property: There are no transitive dependencies in T.
(4) FK property: The root of T is a key, and for each other node u in T, if $FK(D(u)) \neq \emptyset$, then $u \in FK(D(u))$.

In the FK property, ***key*** refers to $LHS(M')$.

We will prove these four properties hold after each major step of Algorithm 1 in which scheme trees are created or modified.

Proposition 7.1: The four properties of NNF hold after step 2 of Algorithm 1 where the initial scheme trees are created.

Proof:

(1) Inference property: Since all MVDs and EMVDs in $MVD(T)$ are trivial for the single path trees which procedure DECOMP produces, this property holds trivially.

(2) PD property: Assume there is a partial dependency in T. Then the path set of T can be decomposed using the partial dependency, and therefore is not in 4NF. This contradicts the fact that we start with all path sets being in 4NF as a result of step 1 of Algorithm D.

(3) TD property: Trivially true, since the definition of transitive dependency requires sibling nodes to exist in the tree, and there are none in a single path tree.

(4) FK property: Procedure DECOMP creates non-leaf nodes which are fundamental keys of the subtrees with those nodes as root. Thus, this property holds by design. □

Proposition 7.2: The four properties of NNF hold after step 3 of Algorithm 1 where scheme trees are merged.

Proof: Assume there are m trees $T_1, T_2, \ldots, T_m$ at some stage of step 3. We show that each property holds after two trees T_1 and T_2 are permanently merged into tree T'.

(1) Inference property: Figure 7-1 shows two general trees T_1 and T_2 with common attributes $u_1, u_2, \ldots, u_n$ forming the same root to non-leaf path in both trees. Subtrees are summarized by the union of all nodes in the subtree (e.g., Y_1^2, Z_1^1). Each of these trees is assumed to be in NNF. Given that these trees are in NNF and the path sets are in 4NF, the following JD holds:

$$\begin{aligned}\bowtie (&u_1Z_1^1, \ldots, u_1Z_{k_1}^1, u_1u_2Z_1^2, \ldots, u_1u_2Z_{k_2}^2, \ldots, u_1u_2\cdots u_nZ_1^n, \ldots, u_1u_2\cdots u_nZ_{k_n}^n,\\ &u_1Y_1^1, \ldots, u_1Y_{j_1}^1, u_1u_2Y_1^2, \ldots, u_1u_2Y_{j_2}^2, \ldots, u_1u_2\cdots u_nY_1^n, \ldots, u_1u_2\cdots u_nY_{j_n}^n,\\ &S(T_3), \ldots, S(T_m)).\end{aligned}$$

This JD implies

$$u_1 \longrightarrow\!\!\!\!\!\rightarrow S(u_2)|Y_1^1|\cdots|Y_{j_1}^1|Z_1^1|\cdots|Z_{k_1}^1$$

holds in T' which is the EMVD representing edge (u_1, u_2). Similarly, the JD implies each edge (u_i, u_ℓ), $1 \leq i = \ell - 1 \leq n-1$. Also, the EMVDs represented by each edge in the Y and Z subtrees are still implied by this JD. Therefore, $MVD(T')$ holds and the inference property is maintained.

(2) PD property: Holds as in Proposition 8.1, since we have not modified the 4NF path sets by merging T_1 and T_2.

(3) TD property: By design, we specifically test that this property is not violated before we merge trees permanently.

(4) FK property: Even though some of the non-leaf u_i nodes may have a new set of descendants consisting of the Y^i and Z^i subtrees at that level, u_i will still be a fundamental key of $V = u_iY_1^iY_2^i\cdots Y_{j_i}^iZ_1^iZ_2^i\cdots Z_{k_i}^i$. If u_i was a minimal intersection of a subset of V and the keys of M', then it will be minimal for V. □

By Propositions 7.1 and 7.2, we know that relations designed using Algorithm 1 will be in nested normal form with respect to M' and F. Since $D \Rightarrow M'$, and $M' \cup F \Rightarrow MVD(T)$, we have a good representation of the original dependencies in our design.

8. Further Normalization of NNF Relations

Algorithm 1 produces a set of ¬1NF relations which is in NNF with respect to a set of MVDs (M') and a set of EMVDs (F). M' was derived from a set D of MVDs and FDs in step one of the algorithm. Later steps deal only with M' and F, and ignore the FDs that existed in D. This was appropriate since we incorporated the FDs by closing the left hand sides of the MVDs in D to obtain M'. This eliminates the possibility of

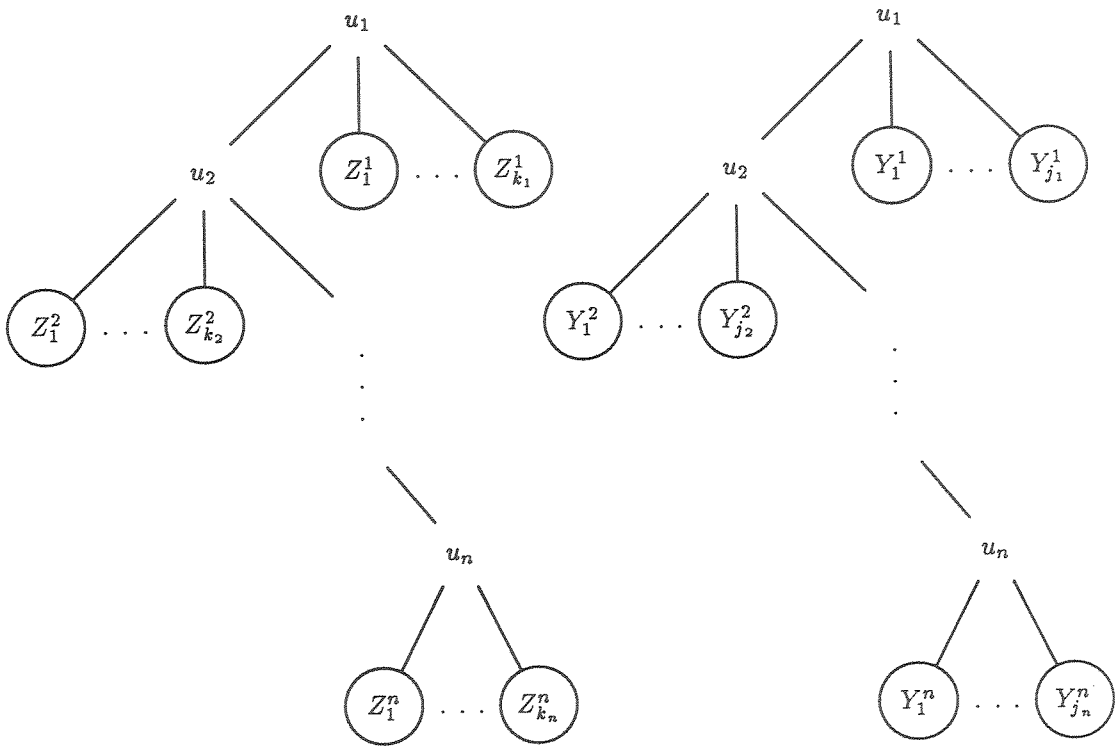


Figure 7-1. General trees T_1 and T_2.

getting nested relations which will only have a single tuple in them. However, there is still a place where redundancy due to FDs arises in our ¬1NF design. The following example will illustrate this problem.

Example 8.1: Consider the following university database taken from [Lie]. We have attributes Class, Day, Hour, Tutor, Office, Student, Major, and Exam. The dependencies which hold are

$$\text{Class} \twoheadrightarrow \text{Day}$$
$$\text{Student} \rightarrow \text{Major}$$
$$\text{Tutor} \rightarrow \text{Office}$$
$$\text{Class, Student} \twoheadrightarrow \text{Exam}$$
$$\text{Class, Tutor} \twoheadrightarrow \text{Hour}$$

Following Algorithm 1, we make the MVDs full and close their left hand sides giving M':

$$\text{Class} \twoheadrightarrow \text{Day} \mid \text{Hour, Tutor, Office, Student, Major, Exam}$$
$$\text{Class, Student, Major} \twoheadrightarrow \text{Exam} \mid \text{Day} \mid \text{Hour, Tutor, Office}$$
$$\text{Class, Tutor, Office} \twoheadrightarrow \text{Hour} \mid \text{Day} \mid \text{Exam, Student, Major}$$

The only 4NF decomposition consists of the following five schemes:

Class, Day

Class, Student, Major, Exam

Class, Tutor, Office, Hour

Class, Tutor, Office, Student, Major

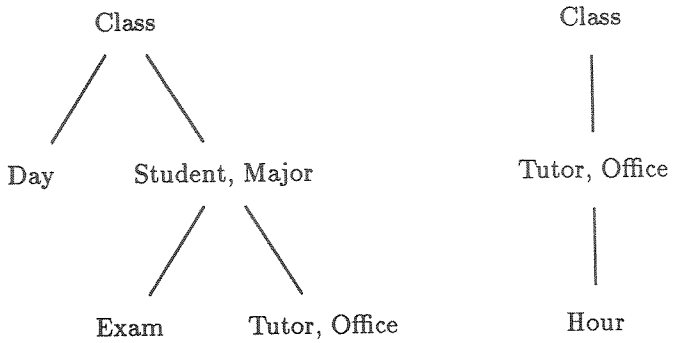


Figure 8-1. Scheme trees produced by Algorithm 1 for Example 8.1.

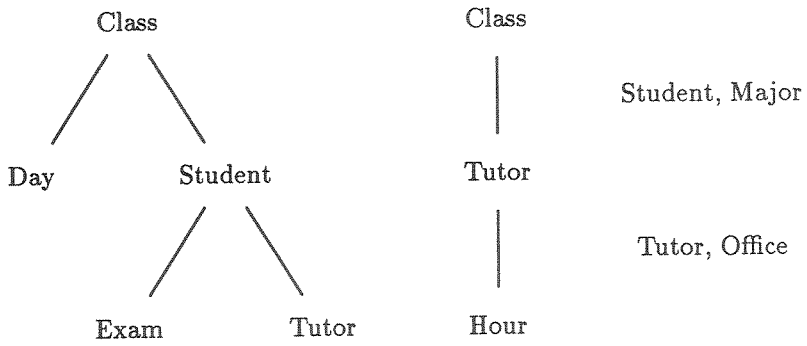


Figure 8-2. Scheme trees produced by Algorithm 2 for Example 8.1.

One of the two symmetric choices that Algorithm 1 produces for this set of schemes is shown in Figure 8-1.

Although this design is in NNF with respect to M', there are some obvious redundancies involving the nodes Tutor, Office and Student, Major. Since the FD Student $\rightarrow$ Major holds, each time a Student value is repeated for different Class values, the same Major value is also repeated. The situation is worse for Tutor, Office. Since the FD Tutor $\rightarrow$ Office holds, each time a Tutor value is repeated for different Class and Student values in one relation, and for different Class values in the other relation, the same Office value is also repeated. □

The problem with designs, such as those in this example, is that groups of attributes which are nested will introduce redundancies if a subset of that group functionally determines some other part of the group. Note that this problem does not occur if the group is at the root of the scheme tree, since these are the atomic attributes of the relation and their values will occur only once in the relation.

The solution is to examine each scheme tree produced by Algorithm 1 for nodes N that exhibit the above behavior, replacing each N with the smallest set of attributes which functionally determines N, and creating a new relation with a single node containing the attributes involved in the redundant FD as root and no branches. For Example 8.1, the new design would consist of the four scheme trees shown in Figure 8-2. Below we give a new algorithm to implement these changes.

It is straightforward to show that the scheme trees produced by Algorithm 2 are still in NNF with respect to M', and if we consider the FDs used to change the trees then the path sets are still in 4NF, and so the join dependency among the path sets continues to hold.

Algorithm 2
{ input: a set $\mathcal{T}$ of scheme trees produced by Algorithm 1
and the set of FDs G used as input to Algorithm 1
output: a new set $\mathcal{T}$ of scheme trees in NNF
with redundancies due to FDs removed. }
begin
Until no more changes can be made to $\mathcal{T}$ do
begin
If there exists $T \in \mathcal{T}$, where T contains a node N such that
$X \rightarrow N$ is implied by G, with $X \subset N$ then
begin
(a) Let $Z \subset N$ where $G \Rightarrow Z \rightarrow N$, and
for no $W \subset Z$ does $G \Rightarrow W \rightarrow N$.
(b) Let $Y \subset Z$ where $G \Rightarrow Y \rightarrow N - (Z - Y)$, and
for no $W \subset Y$ does $G \Rightarrow W \rightarrow N - (Z - Y)$.
(c) Modify T by replacing node N with Z.
(d) Add a new tree to $\mathcal{T}$ with the single node $N - (Z - Y)$ and no edges.
end
end
end.

9. Conclusion

We have given new algorithms for nested normal form which take into account the different semantics of functional and embedded multivalued dependencies, while preserving the good properties of the original nested normal form proposed in [OY]. Furthermore, by utilizing an initial decomposition into fourth normal form, we believe the database designer is given more control over the objects and relationships to be emphasized in the final hierarchical design.

10. Acknowledgements

The authors thank Meral Özsoyoğlu for many helpful discussions on this exposition.

11. References

[AB] Abiteboul, S. and N. Bidoit, "Non First Normal Form Relations to Represent Hierarchically Organized Data," *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Waterloo (April 1984), 191–200.

[AC] Arora, A. and C. Carlson, "On the Flexibility Provided by Conflict-Free Normalization," *Proceedings of the 6th International Computer Software Applications Conference*, Chicago (November 1982), 202–206.

[BaKi] Batory, D. and W. Kim, "Modeling Concepts for VLSI CAD Objects," *ACM Transactions on Database Systems 10*, 3 (September 1985), 322–346.

[BFMY] Beeri, C., R. Fagin, D. Maier and M. Yannakakis, "On the Desirability of Acyclic Database Schemes," *Journal of the ACM 30*, 3 (July 1983), 479–513.

[BeK1] Beeri, C. and M. Kifer, "Comprehensive Approach to the Design of Relational Database Schemes," *Proceedings of the Tenth International Conference on Very Large Databases*, Singapore (August 1984), 196–207.

[BeK2] Beeri, C. and M. Kifer, "Elimination of Intersection Anomalies From Database Schemes," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Atlanta (March 1983), 340–351.

[D+] Dadam, P., K. Kuespert, F. Andersen, H. Blanken, R. Erbe, J. Guenauer, V. Lum, P. Pistor and G. Walch, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat

Tables and Hierarchies," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Washington (1986), 356–367.

[Fag] Fagin, R., "Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systems 2*, 3 (September 1977), 262–278.

[FT] Fischer, P. and S. Thomas, "Operators for Non-First-Normal-Form Relations," *Proceedings of the 7th International Computer Software Applications Conference*, Chicago (November 1983), 464–475.

[FV] Fischer, P. and D. Van Gucht, "Determining when a Structure is a Nested Relation," *Proceedings of the Eleventh International Conference on Very Large Databases*, Stockholm (August 1985), 171–180.

[GR] Grahne, G. and K. Räihä, "Database Decomposition into Fourth Normal Form," *Proceedings of the Ninth International Conference on Very Large Databases*, Florence (October 1983), 186–196.

[JS] Jaeschke, G. and H. Schek, "Remarks on the Algebra of Non First Normal Form Relations," *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Los Angeles (March 1982), 124–138.

[KTT] Kambayashi, Y., K. Tanaka and K. Takeda, "Synthesis of Unnormalized Relations Incorporating More Meaning," *Information Sciences 29* (1983), 201–247.

[Kat] Katsuno, H., "An Extension of Conflict-Free Multivalued Dependencies," *ACM Transactions on Database Systems 9*, 2 (June 1984), 309–326.

[Lie] Lien, Y., "Hierarchical Schemata for Relational Databases," *ACM Transactions on Database Systems 6*, 1 (March 1981), 48–69.

[LT] Loizou, G. and P. Thanisch, "Do Embedded Dependencies Always Lead to a Wild Goose Chase?," *Proceedings of the Third British National Conference on Databases (BNCOD3)*, Leeds (July 1984), 41–56.

[Mai] Maier, D., *Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983.

[Mak] Makinouchi, A., "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model," *Proceedings of the Third International Conference on Very Large Databases*, Tokyo (October 1977), 447–453.

[OO] Özsoyoğlu, G. and Z. Özsoyoğlu, "SSDB–An Architecture for Statistical Databases," *Proceedings of the 4th Jerusalem Conference on Information Technology*, Jerusalem (May 1984), 327–341.

[OY] Özsoyoğlu, Z. and L. Yuan, "A Normal Form for Nested Relations," *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Portland (March 1985), 251–260.

[Ozs] Özsoyoğlu, Z., Personal communication, 1986.

[PP] Parker, D. and K. Parsaye-Ghomi, "Inferences Involving Embedded Multivalued Dependencies and Transitive Dependencies," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Santa Monica (May 1980), 52–57.

[RKS] Roth, M., H. Korth and A. Silberschatz, "Extended Algebra and Calculus for ¬1NF Relational Databases," TR-84-36, Department of Computer Science, University of Texas at Austin (December 1984), revised January 1986.

[Sac] Sacca, D., "On the Recognition of Coverings of Acyclic Database Hypergraphs," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Atlanta (March 1983), 297–304.

[Sch1] Schek, H.-J., "Methods for the administration of textual data in database systems." In *Information Retrieval Research*, Oddy, Robinson, Van Rijsbergen and Williams, Eds., Buttersworth, London, 1981, 218–235.

[Sch2] Schek, H.-J., "Towards a Basic Relational NF^2 Algebra Processor," *International Conference on Foundations of Data Organization*, Kyoto, Japan (May 1985), 173–182.

[SP] Schek, H.-J. and P. Pistor, "Data Structures for an Integrated Data Base Management and Information Retrieval System," *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City (September 1982), 197–207.

[Sci1] Sciore, E., "Improving Database Schemes by Adding Attributes," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Atlanta, (March 1983), 379–383.

[Sci2] Sciore, E., "Real-World MVD's," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Ann Arbor (April 1981), 121–132.

[Sci3] Sciore, E., "Some Observations on Real-World Data Dependencies." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.

[SLTC] Shu, N., Y. Lum, F. Tung and C. Chang, "Specification of Forms Processing and Business Procedures for Office Automation," *IEEE Transactions on Software Engineering 8*, 5 (September 1982), 499–512.

[Ull] Ullman, J., *Principles of Database Systems*, 2nd Edition, Computer Science Press, Potomac, MD, 1982.

[YO] Yuan, L. and Z. Özsoyoğlu, "Unifying Functional and Multivalued Dependencies for Relational Database Design," *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*,Cambridge (March 1986), 183–190.