INFORMATION SCIENCES **52**, 53–73 (1990)

The Synthesis Approach for Relational Database Design: An Expanded Perspective

SUDHA RAM

and

STEPHEN M. CURRAN

Department of Management Information Systems,
College of Business and Public Administration, University of Arizona, Tucson, Arizona 85721

ABSTRACT

This paper addresses relational database design using the concept of functional dependencies (FDs). The classical synthesis approach processes a given set of functional dependencies to produce one minimal cover. This cover is then used to develop a relational schema; however, a given set of FDs may have more than one minimal cover. In turn, different minimal covers may give rise to different relational schemata. An enhancement is proposed to the traditional synthesis algorithm that aids in efficiently determining *all* minimal covers for a given set of functional dependencies. The algorithm has been implemented using Turbo Pascal on an IBM PC/AT. The performance of this algorithm is compared with that of the traditional synthesis algorithm.

1. INTRODUCTION

The database design process can be divided into four stages. These stages are as follows:

(1) *Requirements collection*: Users are asked about their needs.
(2) *Conceptual design*: A conceptual model of the application is developed using the data collected in the first stage. This model is independent of the database management system to be employed.
(3) *Logical design*: The conceptual model is translated into a specific model such as relational, network, or hierarchical. For instance, if a relational system is to be used, the design of the relations is specified.
(4) *Physical design*: The appropriate storage structures, indexes, etc., are selected.

0020-0255/90/$03.50

This paper presents an algorithm that can be applied during the logical design of relational databases. The algorithm provides database design alternatives that are not readily produced by current manual or automated design techniques. The next section clarifies the terminology and notation used in this paper and briefly describes two major approaches to relational design. The same section then concentrates on describing the classical synthesis approach and justifies the need for a modified synthesis approach. Section 3 describes a modified synthesis algorithm. Performance benchmarks for the operational algorithm are described in Section 4. Comparisons with the classical synthesis approach [3] are reported as well. Section 5 concludes the paper by identifying directions for future research.

The reader is assumed to be familiar with the theory of functional dependencies and concept of normal forms up to the level of 3NF [6].

2. MAJOR APPROACHES FOR RELATIONAL DESIGN

2.1. *TERMINOLOGY AND NOTATION*

In this paper, we will use the notation adopted in Ullman [11]. X, Y and Z will be used to denote sets of attributes (one or more attributes), while A, B, C will be used to denote individual attributes. We will restrict the class of data dependencies considered to functional dependencies. Multivalued dependencies (MVDs) are not addressed. An FD, represented as $X \rightarrow Y$, describes a logical association between two sets of attributes, X and Y. Thus, FDs act as structural descriptors as well as integrity constraints. In some cases an FD will also be referred to by the symbol $f1$ or $f2$. Lowercase letters will be used when generic example sets of FDs are considered. The closure of a set of attributes X with respect to a set of FDs F will be denoted by $(X_F)^+$. It is the set of all attributes derived from X using the FDs in F and Armstrong's axioms [9–11]. F, G, H, and H' will be used to denote sets of functional dependencies.

Given a set of FDs, it is possible that some are implied by others. As will be discussed in the next sections, Bernstein's synthesis approach tries to eliminate all such FDs. The final set of FDs thus derived constitutes a minimal cover, which is then used for synthesizing relations in 3NF. The concept of minimal cover is important in this context. A minimal cover for a set of FDs is defined as follows [11]:

[1] Every right side of a dependency in F is a single attribute.

[2] For no $X \rightarrow A$ in F is the set $F - \{X \rightarrow A]$ equivalent to F.

[3] For no $X \rightarrow A$ in F and proper subset Z of X is $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ equivalent to F.

Intuitively, [2] guarantees that no dependency in F is redundant, and [3] guarantees that no attribute on any left side is redundant. Since each right side has a single attribute (as implied by [1]), no attribute on the right is redundant.

At this point we would like to distinguish between the terms *implied* extraneous attributes and *nonimplied* extraneous attributes. If X is the LHS of an FD and $B \in X$ and $B \in (X - B)_F^\dagger$, then we call B an *implied* extraneous attribute. Other extraneous attributes are *nonimplied*.

2.2. DECOMPOSITION AND SYNTHESIS

The concept of *normal forms* has provided a sound basis for relational design [1,4,5]. Identifying logical associations among attributes is a prerequisite for designing relations. These logical associations, also known as *functional dependencies* (FD), can be used in two different ways to yield an effective and efficient design. The two approaches are:

(1) decomposition,
(2) synthesis.

The synthesis approach was pioneered by Bernstein [3] and the decomposition approach was proposed by Codd and later generalized by Fagin [8]. The advantages and disadvantages of the two approaches are described very well in Maier [10], a brief summary of which is provided here.

The decomposition approach uses a set of relations that are not in 3NF with respect to a set of FDs (F) and decomposes them into 3NF relations by removing anomalies. This approach is particularly useful in the presence of MVDs; however, it has three disadvantages. First, the process does not have polynomial time complexity. Further, it often yields more relations than are actually needed. Lastly, it may produce a design that does not enforce some of the FDs in F. To avoid these problems, the synthesis approach can be used.

In the synthesis approach, relations are created directly from the set of FDs. The core of the problem lies in determining the proper set of FDs to use. In general, this approach works well for FDs, but is not suitable for processing MVDs. The synthesis approach has been enhanced by several authors [12,13]. Some authors [2] have suggested a combination of the decomposition and synthesis approaches while considering a restricted class of FDs, thus allowing both FDs and MVDs.

2.3. THE CLASSICAL SYNTHESIS APPROACH

A brief summary of the traditional synthesis approach is presented in this section before we suggest an enhancement.

The synthesis algorithm is summarized below:

ALGORITHM 2.3.1.

Step 1: *Eliminate extraneous attributes.* Let F be a given set of FDs. Eliminate extraneous attributes from the left side of each FD in F, producing the set G. An attribute is extraneous if its elimination does not alter the closure of the set of FDs.

Step 2: *Find covering.* Find a nonredundant covering H of G, viz., eliminate redundant FDs from G.

Step 3: *Partition.* Partition H into groups such that all of the FDs in each group have identical left hand sides.

Step 4: *Construct relations.* For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation. The set of relations constructed constitutes a schema for a given set of FDs.

2.4. *MOTIVATION FOR MODIFIED BERNSTEIN ALGORITHM*

The cover that is produced at the end of step 2 is referred to as a *minimal cover*. It has been proved that Algorithm 2.3.1 produces relations in 3NF [3]. It is generally agreed that to perform this synthesis only one minimal cover is required; however, a given set of FDs may have several minimal covers. In turn, differences between minimal covers may in fact produce different relation schemes. This is best illustrated by an example:

EXAMPLE 2.4.1. Consider a set of attributes—Course (c), Teacher (t), Hour (h), Student (s), Grade (g), Room (r)—with the following FDs [11]:

$$\begin{array}{ll} c \rightarrow t & hr \rightarrow c \\ th \rightarrow r & cs \rightarrow g \\ hs \rightarrow r & hs \rightarrow c \end{array}$$

Using steps 1 and 2 of Algorithm 2.3.1, we can identify two minimal covers for this set of FDs:

Cover 1	Cover 2
$c \rightarrow t$	$c \rightarrow t$
$th \rightarrow r$	$th \rightarrow r$
$hr \rightarrow c$	$hr \rightarrow c$
$cs \rightarrow g$	$cs \rightarrow g$
$hs \rightarrow c$	$hs \rightarrow r$

The two covers differ by one FD, namely the last one. $hs \rightarrow c$ in cover 1 implies that students cannot attend two different *courses* during the same hour. $hs \rightarrow r$ implies that students cannot occupy two different *rooms* during the same hour. Depending on the minimal cover chosen, the relational schema will vary. While it has been proved that all minimal covers produce the same number of relations (minimal number) [3], these relations will depend on the minimal cover chosen for generating the schema. For instance in the example above the two relational schemata will be:

Schema 1	Schema 2
(Course, Teacher)	(Course, Teacher)
(Teacher, Hour, Room)	(Teacher, Hour, Room)
(Hour, Room, Course)	(Hour, Room, Course)
(Course, Student, Grade)	(Course, Student, Grade)
(Hour, Student, Course)	(Hour, Student, Room)

The two derived schemata produce the same number of relations; however, they differ by one relation. Given this choise, the selection of a suitable schema depends on the processing requirements for the application. For example, the *best* schema should be able to answer queries as efficiently and effectively as possible. If schema 2 is chosen, queries pertaining to Hour, Student, and Room can be processed without joins. In contrast, if queries predominantly pertain to Hour, Student, and Course, at least one join has to be performed for each query. In this instance, schema 1 would have been a better choice. Clearly, generating all minimal covers provides an opportunity to select the most effective and efficient relational schema for a given application.

Most synthesis approaches produce just one minimal cover and use it to generate the relations in 3NF; however, the order in which FDs are submitted to Algorithm 2.3.1 may affect the minimal cover produced. Distinct covers can be produced by reordering the FDs. In order to find *all* minimal covers, the FDs have to be reordered in all possible distinct ways. This is extremely inefficient. In the next section we will describe a more effective approach to find all minimal covers. This section will also describe heuristics that can be applied to select the minimal cover that best meets processing requirements.

3. A MODIFIED ALGORITHM TO DETERMINE MINIMAL COVERS

This section describes an alternative approach to determining all minimal covers for a given set of FDs. All the modifications are made to step 2 of

Algorithm 2.3.1. The third subsection describes the heuristics used for choosing a minimal cover.

3.1. THE BASIC ALGORITHM

Given a set of functional dependencies (F), the first step in the algorithm eliminates all extraneous attributes as described by Bernstein [3]. During the first step of the algorithm, implied extraneous attributes are removed. While the resulting set of FDs (G) no longer contains implied extraneous attributes, it may contain nonimplied extraneous attributes. Further, some of the FDs may also be duplicates. During step 2, duplicate FDs and FDs with nonimplied extraneous attributes are removed from G. Next, each of the FDs in G is considered to determine if it is redundant. This step is conducted in the standard manner. Consider each FD $X \to A$ in G. If the set $G - \{X \to A\}$ is equivalent to G, then $X \to A$ is considered redundant.

In Bernstein's algorithm, once a given FD ($f1$) is found to be redundant, it is eliminated from the list G. In our algorithm, we continue to retain the FD in G; however, we flag $f1$ as being redundant. Step 2 culminates by partitioning the set of FDs (G) into two subsets: one containing the FDs that are not redundant, and the other containing the FDs that are potentially redundant. Let us refer to the former set as the *basic set* (H) and the latter as the *evaluation set* (H').

In the next step each potentially redundant FD from the set H' is examined to determine its source of redundancy. For example, the potentially redundant FD $X \to A$ is removed from G. Now reconsider every remaining potentially redundant FD ($Y \to B$) in the subset H' of G for redundancy. If every $Y \to B$ remains redundant, then $X \to A$ can be removed from the original set of FDs (G) without any loss of information, because it represents a transitive dependency. In contrast, if any $Y \to B$ in H' is found to be nonredundant, then it can be inferred that $X \to A$ affects $Y \to B$ and should be temporarily retained in G for further evaluation. Thus, the interaction between $X \to A$ and $Y \to B$ indicates that removal of the FD $X \to A$ from G caused $Y \to B$ to be nonredundant in G. Similarly, if removal of $Y \to B$ causes $X \to A$ to be nonredundant, then $X \to A$ and $Y \to B$ can be regarded as *perfect substitutes*. The determination of this set of perfect substitutes implies either $f1$ or $f2$ ($f1$ and $f2$ refer to two different FDs) is required in the minimal cover. If removal of an FD $X \to A$ does not have any effect on the remaining FDs in H', then $X \to A$ is identified as being totally redundant and can be deleted from the set G. It is also possible that removal of the FD $X \to A$ from the set G causes more than one of the other FDs in H' to become nonredundant. To retain information about the source of redundancy, we create sets of FDs of the type $\{M:\{R\}\}$, where M refers to the FD that is being considered from the set H', and $\{R\}$ refers to the set of one or more FDs that are no longer redundant because of the removal of the FD M. Using the $\{M:\{R\}\}$ sets, we can generate several minimal covers that are presented to the user. The user may then select the cover that suits his purposes best. In case the user is not able to make a choice, it may be possible to apply several heuristics to choose a suitable minimal cover. A description of these heuristics is given at the end of this section. We are currently building a prototype that will incorporate them to determine the minimal cover that is most suitable for the user.

3.2. *EXTENSIONS TO THE ALGORITHM*

Unfortunately, the approach given above does not deal effectively with special cases of FDs. These special cases are identified and handled by introducing additional steps to our algorithm.

The first special case occurs when the FDs are of the following type:

$$a1 \rightarrow b1\,c1\,d1\,e1\,f1$$

$$b1 \rightarrow a1\,c1\,d1\,e1\,f1$$

These FDs indicate that $a1$ and $b1$ can be regarded as alternative keys. Rather than taking a permutation of the FDs containing either $a1$ or $b1$ on the left hand side, we partition the FDs into two sets of which one contains only the FDs with $a1$ on the left hand side and the other contains only those with $b1$ on the left hand side. One of these two sets is removed from H' and inserted into H. The other set is eliminated from further consideration without causing any loss of information.

The other special case arises when groups of independent FDs are submitted together to the algorithm. This is the case when the sets of FDs correspond to different applications. In such cases, the $\{M:\{R\}\}$ sets must be partitioned into mutually independent sets in order to derive the minimal covers. This procedure is explained in detail in Example 3.2.1. The following procedure is a summary of the algorithm described above:

ALGORITHM 3.2.1

Input: A set of FDs (F).
Output: 3NF relations.

Step 1: *Eliminate extraneous attributes.* Eliminate extraneous attributes from the left side of each FD in F, producing the set G. This step is identical to step 1 of Algorithm 2.3.1. An attribute is extraneous if its elimination does not affect the closure of the set of FDs.

Step 2: *Check for redundant FDs.* Delete duplicate FDs; delete FDs that have *nonimplied* extraneous attributes on the LHS; mark all the

FDs that are determined to be potentially redundant. Partition G into two sets H and H'. H is the basic cover that contains all nonredundant FDs, and H' contains FDs that are found to be potentially redundant.

Step 3: *Determine alternative key FDs.* The LHS attributes that can be regarded as alternative key attributes are determined. Using these attributes, H' is partitioned into mutually exclusive sets $H1', H2', \ldots$. One of these sets is added to the basic set, while all the others are removed from H'.

Step 4a: *Create* $\{M:\{R\}\}$ *sets.* Consider the set H' which consists of potentially redundant FDs. Consider an FD, $X \to A$, from the set H'. Determine the effect of removing $X \to A$ from the set G on the redundancy for all other FDs of H', thus creating the $\{M:\{R\}\}$ sets. If removal of $X \to A$ has no effect on any of the remaining FDs in H', remove it from G.

Step 4b: *Partition the* $\{M:\{R\}\}$ *sets.* Once all the $\{M:\{R\}\}$ sets are identified, they are partitioned into mutually independent sets using directed graphs. This procedure is explained in detail in Example 3.2.1.

Step 4c: *Choose one minimal cover.* From each independent cluster of $\{M:\{R\}\}$ sets choose one FD belonging to the M type. Use heuristics (described in Section 3.3) for making the choice. The FDs thus chosen are added to the basic set H, thus creating a minimal cover.

Step 5: *Partition.* Partition the minimal cover into groups such that all of the FDs in each group have identical left hand sides.

Step 6: *Construct relations.* For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation. The set of relations constructed constitutes a schema for a given set of FDs.

Algorithm 3.2.1 can be implemented to have a time complexity of $O(n^2)$, where n is the number of FDs submitted.

An illustration of this algorithm is given below:

EXAMPLE 3.2.1. Consider the following set of FDs [11]:

$$\begin{array}{rr} ab \to c & cg \to d \\ bc \to d & ce \to g \\ acd \to b & ce \to a \\ be \to c & c \to a \\ cg \to b & d \to e \\ & d \to g \end{array}$$

Step 1: Consider each FD, and remove extraneous attributes. Clearly $acd \rightarrow b$ can be replaced by $cd \rightarrow b$, since a is an extraneous attribute. No other FDs in this set contain extraneous attributes. Thus the set of FDs (G) is now

$$\begin{array}{ll} ab \rightarrow c & ce \rightarrow g \\ bc \rightarrow d & ce \rightarrow a \\ cd \rightarrow b & c \rightarrow a \\ be \rightarrow c & d \rightarrow e \\ cg \rightarrow b & d \rightarrow g \\ cg \rightarrow d & \end{array}$$

Step 2: Check for redundant FDs. Note that there are no duplicate FDs in the above set after the extraneous attributes were removed as shown in step 1; however, $ce \rightarrow a$ is a redundant FD, since it contains a nonimplied extraneous attribute on the LHS. Thus, $ce \rightarrow a$ is removed from the set G. No other FDs are found to contain nonimplied extraneous attributes. Now we proceed to mark all the potentially redundant FDs. For instance consider the FD $cd \rightarrow b$. Clearly it is redundant because of the existence of $cg \rightarrow b$. In the original synthesis procedure [3], $cd \rightarrow b$ would have been eliminated from G. However, in our modified procedure it is retained. The processing is continued to generate all the other potentially redundant FDs. In this example, after one pass through G, the following potentially redundant FDs are generated for the evaluation set H':

$$\begin{array}{r} cd \rightarrow b \\ cg \rightarrow b \\ cg \rightarrow d \end{array}$$

The basic set H consists of

$$\begin{array}{r} ab \rightarrow c \\ c \rightarrow a \\ bc \rightarrow d \\ d \rightarrow e \\ be \rightarrow c \\ d \rightarrow g \\ ce \rightarrow g \end{array}$$

Step 3: Determine alternative key FDs. Consider the set H. There are no two FDs, $f1$ and $f2$, such that the LHS of each determines the RHS of the other. Hence, H and H' are not modified any further in this step.

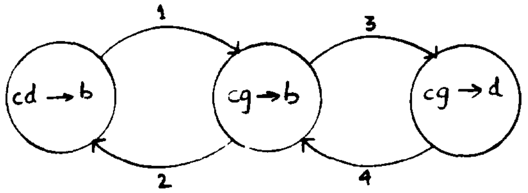

Fig. 1.

Steps 4a, 4b, and 4c: Consider each FD in H'. Remove the FD from the original list of FDs (G), and determine if any of the other members of H' are affected. For example, if $cd \to b$ is removed from G, then $cg \to b$ is no longer redundant but $cg \to d$ is redundant. Hence one $\{M:\{R\}\}$ set would consist of $\{cd \to b:\{cg \to b\}\}$. Similarly, the other $\{M:\{R\}\}$ sets would be

$$\{cg \to b:\{cd \to b, cg \to d\}\},$$

$$\{cg \to d:\{cg \to b\}\}.$$

Each of these $\{M:\{R\}\}$ sets is evaluated using directed graphs to determine if any of them are mutually independent. The FDs in each $\{M:\{R\}\}$ sets are represented by the nodes of the directed graph. Each node of the directed graph represents one FD (see Figure 1).

There is only one node corresponding to each FD in the directed graph, even though an FD may occur in more than one $\{M:\{R\}\}$ set. The nodes are linked to each other by means of directed arcs. Each arc is drawn so that it leaves the node representing the FD that forms the M part of an $\{M:\{R\}\}$ set and enters each of the nodes representing the FDs in the $\{R\}$ part of each $\{M:\{R\}\}$ set. In this example, using the first $\{M:\{R\}\}$ set, arc 1 leaves the node representing the FD $cd \to b$ and enters the node representing the FD $cg \to b$. Similarly, using the second $\{M:\{R\}\}$ set, arcs 2 and 3 leave the node representing the FD $cg \to b$ and enter the nodes representing the FDs $cd \to b$ and $cg \to d$ respectively. Arc 4 represents the third $\{M:\{R\}\}$ set identified above. Independent clusters of $\{M:\{R\}\}$ sets are identified easily using this procedure, since the nodes in each cluster will not be connected by arcs to the nodes in any of the other clusters. In this example none of the $\{M:\{R\}\}$ sets are mutually independent; therefore, partitioning is not required.

One minimal cover would be $H \cup \{cg \to b\}$. When $cg \to b$ is removed from the original set G, both $cd \to b$ and $cg \to d$ are no longer redundant. Thus, another minimal cover consists of $H \cup \{cd \to b, cg \to d\}$. When $cg \to d$ is removed from the set G, $cd \to b$ is redundant, while $cg \to b$ is no longer redundant. Consequently, a third minimal cover would be $H \cup \{cg \to b\}$. Note that it is the same as the first one. By examining the $\{M:\{R\}\}$ sets, it is obvious that $cd \rightarrow b$ and $cg \rightarrow b$ are perfect substitutes, i.e., each causes the other to become redundant.

The following example illustrates the need to eliminate the alternative key FDs and the need to partition the $\{M:\{R\}\}$ sets into mutually independent sets.

EXAMPLE 3.2.2. Consider the following set of FDs adapted from [11, 7]:

Set 1	Set 2		
$c \rightarrow t$	$a \rightarrow b$	$d \rightarrow f$	$i \rightarrow l$
$th \rightarrow r$	$a \rightarrow d$	$d \rightarrow b$	$i \rightarrow m$
$hs \rightarrow r$	$a \rightarrow e$	$d \rightarrow q$	$j \rightarrow i$
$hr \rightarrow c$	$a \rightarrow f$	$q \rightarrow d$	$j \rightarrow k$
$cs \rightarrow g$	$a \rightarrow p$	$i \rightarrow j$	$j \rightarrow l$
$hs \rightarrow c$	$a \rightarrow q$	$i \rightarrow k$	$j \rightarrow m$

It is possible that set 1 and set 2 are submitted together to the synthesis algorithm.

Step 1 (eliminate extraneous attributes): There are no extraneous attributes in the given set of FDs. Thus, set G consists of the original set that was submitted.

Step 2 (check for redundant FDs): There are no duplicate FDs or FDs with *nonimplied* attributes on the LHS; however, there are several FDs that are potentially redundant. After identifying potentially redundant FDs, the set G consists of the following partitions:

Basic set H		Evaluation set H'	
$c \rightarrow t$	$a \rightarrow p$	$hs \rightarrow c$	$i \rightarrow m$
$th \rightarrow r$	$d \rightarrow f$	$hs \rightarrow r$	$j \rightarrow k$
$hr \rightarrow c$	$d \rightarrow b$	$a \rightarrow d$	$j \rightarrow l$
$cs \rightarrow g$	$d \rightarrow q$	$a \rightarrow f$	$j \rightarrow m$
	$a \rightarrow b$	$q \rightarrow d$	$a \rightarrow q$
	$a \rightarrow e$	$i \rightarrow j$	$i \rightarrow k$
	$j \rightarrow i$	$i \rightarrow l$	

Step 3 (remove alternative key FDs): By examining the set H, we determine that the FDs $i \rightarrow j$ and $j \rightarrow i$ are mirror images of each other. Some of the FDs in H' are potentially redundant because of this relationship between i and j. Hence, we remove $j \rightarrow k$, $j \rightarrow l$, and $j \rightarrow m$ from H'. We also move $i \rightarrow k, i \rightarrow l$, and $i \rightarrow m$ to H from H'. Note that it would not make any difference to the final set of relations if we removed $i \rightarrow k$, $i \rightarrow l$, and $i \rightarrow m$

from H' and moved $j \to k, j \to l, j \to m$ from H' to H. Now H' is reduced to the following set:

$$\begin{aligned} hs &\to c \\ hs &\to r \\ a &\to d \\ a &\to f \\ a &\to q \end{aligned}$$

Steps 4a, 4b, 4c (create {M: {R}} sets and generate minimal cover): Consider each FD from the set H'. For instance, remove the first FD in the set H', $hs \to c$. If this FD is removed from the set G, then $hs \to r$ is nonredundant in G; however, none of the other FDs in H' are affected. Similarly, if $hs \to r$ is removed from G, then $hs \to c$ becomes nonredundant in G, while none of the other FDs in H' are affected. By means of similar processing we arrive at the following $\{M:\{R\}\}$ sets:

$$\{hs \to c, \{hs \to r\}\},$$

$$\{hs \to r, \{hs \to c\}\},$$

$$\{a \to d, \{a \to q\}\},$$

$$\{a \to f, \{0\}\},$$

$$\{a \to q, \{a \to d\}\}.$$

The removal of $a \to f$ results in all remaining FDs in H' retaining redundancy. Hence $a \to f$ is determined to be absolutely redundant (transitive dependency) and can be removed from G. From the remaining $\{M:\{R\}\}$ sets is is determined that the first two sets are independent of the remaining two $\{M:\{R\}\}$ sets. Figure 2 illustrates the directed graphs used for arriving at this decision. Note that there are two independent clusters of $\{M:\{R\}\}$ sets with two $\{M:\{R\}\}$ sets in each cluster.

Consequently, the minimal cover consists of the basic set H along with one FD from each of the following partitioned sets:

$$\{hs \to c,\ hs \to r\},$$

$$\{a \to d, a \to q\}.$$

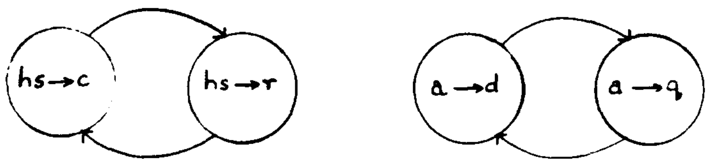

Fig. 2.

LEMMA 3.2.2. *The maximum number of minimal covers is*

$$\prod_{i=1}^{m} n_i,$$

where m is the number of mutually independent partitions and n_i is the number of $\{M:\{R\}\}$ sets in each partition.

Proof. The number of combinations of two partitions with n_1 and n_2 $\{M:\{R\}\}$ sets respectively is $n_1 n_2$. The number of minimal covers produced by each independent partition i is n_i, the number of $\{M:\{R\}\}$ sets in that partition. By induction, the number of combinations of m partitions with n_i $\{M:\{R\}\}$ sets in each is $\prod_{i=1}^{m} n_i$.

Note that some of the minimal covers may be duplicates.

The technique described in this section bypasses the need to reorder FDs and consider all the orders exhaustively. Only the potentially redundant FDs are resubmitted to a redundancy check.

3.3. *HEURISTICS FOR CHOOSING A MINIMAL COVER*

Several simple heuristics can be applied to assist the designer in selecting a minimal cover. As previously mentioned, one plausible selection criterion is to make the relational schema as efficient as possible by minimizing the number of joins required to satisfy application transactions. We will focus on two classes of transactions—queries and updates. Queries refer to transactions that extract data from the database, while updates modify the database by writing data into it.

As described in Algorithm 3.2.1, the $\{M:\{R\}\}$ sets serve to differentiate among the possible minimal covers. The number of queries and updates referencing attributes that compose an FD of type M or the FDs in the set R can provide one basis for selection. For example, if the attributes in a particular set R have more queries and updates referencing them than the attributes in each of the other $\{M:\{R\}\}$ sets, then the former should be selected to create the final minimal cover. In addition, some transactions may be considered to be more important than others. In this case, a weighting factor can be used to indicate the relative importance of each transaction.

In certain cases, the attributes in two or more of the sets may be referenced by almost the same weighted number of queries and updates. In such cases we may follow one of the following strategies:

(1) Choose one set arbitrarily from among the ones that are referenced most often.

(2) Choose to keep in the final cover all of the sets that are referenced most often. Notice that this will introduce a redundancy in the relational schema and could create relations in 2NF.

The choice of strategy is dependent on the transaction type. If the database is to be used most often for querying and not for updating, the second option is better, since joins will be minimized. In contrast, if the database is used most often for updating, the first option is better, since this will reduce the anomalies due to storage of redundant data. Hence there is a tradeoff between redundancy and efficiency defined in terms of joins.

4. BENCHMARKS

In this section we will describe the FD set classes that were processed by Algorithm 3.2.1, and we will also present the benchmark results. We will demonstrate that, while Algorithm 2.3.1 is more efficient than Algorithm 3.2.1 for generating a single minimal cover, our approach efficiently provides information on all possible minimal covers.

We generated six different set classes of FDs. These set classes were modifications of the generic sets described by Diederich and Milton [7]. The basic sets each contain 40–50 FDs. We also created supersets of the basic set classes that contain 80–100, 120–150, and 160–200 FDs, respectively. Finally, we created several composite sets by combining one set class from this section and additional unique sets of FDs. A more detailed description of individual set composition is proved in Appendix A.

The basic sets evaluated represent the following generic classes:

(1) *Clustered dependent* (CDx). This class contains FDs that are linked as a cluster by shared attributes. In addition, a transitional FD links one cluster to another associated cluster of FDs. This class represents descriptors for an employee, department, and office section that form a cluster connected by employee manager, department manager, and employee office. This cluster is

connected to a second cluster for inventory entities by an employee sales representative attribute.

(2) *Clustered independent* (CIx). This is the clustered dependent class with the transitional dependency removed. This class represents a set of FDs with independent clusters.

(3) *Linear reversed* (LRx). This class is a pathological set where each FD represents a unique entity and no redundancies exist.

(4) *Full linear reversed* (FLRx). This class represents a clustered set of FDs that also contains some of the pathology of the linear reversed class.

(5) *Full independent* (FIx). This is the full linear reversed class with FDs that form cluster connections removed. It represents sets with more fully independent groupings of FDs.

(6) *Independent equivalent* (IEx). This class represents sets of FDs that contain equivalent or alternative keys that generate redundant dependencies.

The benchmarks were conducted on an IBM PC/AT using Turbo Pascal 3.0 to implement both Algorithm 2.3.1 and Algorithm 3.2.1. The data structures required to operationalize Algorithm 2.3.1 were designed first in a standalone effort. We believe that the design employed provides an efficient assessment of the performance for this algorithm. The data structures to operationalize Algorithm 3.2.1 required minor adaptations to our initial design for the Bernstein algorithm.

The tables in Appendix B summarize the performance of Algorithm 3.2.1 for each class of FDs submitted. The number of FDs submitted, the number of unique minimal covers generated, and the time in seconds required for each phase of processing are provided for the sets evaluated. The total time reported for each class represents the average of the total times, while the phase times reported represent the averages of the individual phase times. In addition, the time required to process a single minimal cover using the unmodified Bernstein algorithm is provided for comparison. The benchmark total times represent the time required to process steps 1–5 of Algorithm 3.2.1 to determine the minimal cover(s) for a set of FDs.

Algorithm 2.3.1 was technically more efficient than Algorithm 3.2.1 for each class of dependencies submitted. This difference in efficiency arises because the Bernstein algorithm eliminates redundant FDs as soon as they are detected. As FDs are removed, fewer FDs must be processed during redundancy checking. In contrast, Algorithm 3.2.1 initially retains all FDs that have been identified as potentially redundant and only deletes duplicate FDs or FDs with nonimplied extraneous attributes. Additional processing is also required to explicitly identify the source of redundancy and create the $M:R$ sets.

While the unmodified Bernstein algorithm was more efficient in generating a single cover for every case, the lack of apparent efficiency for Algorithm

3.2.1 should be balanced against its potential to provide beneficial design alternatives. First, we can assume that the number of potential minimal covers that will emerge from the synthesis approach is not known in advanced. And if information about additional minimal covers is desired, the effort required to generate all possible covers for a given set of functional dependencies will result in a combinatorial exercise. This problem has combinatorial complexity because algorithm 2.3.1 does not preserve information on the rationale for deleting a given functional dependency. For example, a given functional dependency could be deleted because it described a transistive dependency or because it was simply implied by other combinations of FDs. Thus, to accurately determine all minimal covers using Algorithm 2.3.1, all possible orderings of functional dependencies must be considered. The overhead required to accomplish the reordering and the time required to process each net set could prove to be prohibitive. Even if this were undertaken, the result would be the creation of many duplicate covers.

One alternative approach to reducing this combinatorial complexity is to assign a weighting factor that indicates the relative significance of each FD to the application being modeled. By sorting the list of FDs so that higher priority FDs are placed at the end of the list, we may be able to preserve these more significant FDs from inadvertent deletion.

This alternative offers only a partial solution for several reasons. First, the weighting factor is arbitrarily selected and assigned without full knowledge of the subtle interrelationships that can exist between FDs. As a result, two or more FDs with equal weights may still exist as $M:R$ sets. Again, the FD that emerges for inclusion in the final minimal cover will be selected on the basis more of ordering than of true merit. Ultimately, the designer still loses the opportunity to specify FDs that should be included for the construction of the final relations.

In contrast, Algorithm 3.2.1 not only minimizes the combinatorial problem of determining all plausible, unique minimal covers for a given set of FDs, but also creates information bearing entities that express the interrelationships between redundant FDs. Ultimately, database designers benefit because they can select a minimal cover and synthesize relations that best satisfy the database user's needs.

5. CONCLUSION

We have approached the problem of relational database design using a modified synthesis algorithm. Our algorithm enhances the traditional synthesis algorithm proposed by Bernstein [3]. The modified algorithm has been implemented in Turbo Pascal on an IBM PC/AT with a time complexity of $O(n^2)$.

This algorithm takes 10–80% more time than Bernstein's algorithm; however, it should be noted that instead of producing just one minimal cover, our algorithm produces all minimal covers for a given set of FDs.

Currently we are building a tool that uses this synthesis approach to produce all minimal covers. The tool also incorporates heuristics to choose one minimal cover, and it produces the final relational schema in 3NF and tests it for lossless joins. We also intend to enhance the algorithm to include multivalued dependencies. This will assist in designing relations in 4NF.

APPENDIX A. DESCRIPTION OF BENCHMARK SETS

CD1: *Clustered dependent.*

First cluster:

$$\begin{aligned} &A01 \to B01, C01, D01, E01, F01, G01, H01.\\ &E01 \to J01, K01, M01, N01, P01, R01, S01.\\ &P01 \to D01, T01, U01, V01, H01, W01.\end{aligned}$$

Transitional: $W01 \to A02$.

Second cluster: Same as first cluster with each 01 changed to 02.

CI1: *Clustered independent.* This is CD1 with the transitional dependency removed.

LR1: *Linear reversed.*

$$A01 \to A00,\ A02 \to A01,\ A03 \to A02, \ldots,\ A50 \to A49.$$

FLR1: *Full linear reversed.*

$$\begin{aligned} &A00 \to A01, A02, A03, A04.\\ &B00 \to A00, B01, B02, B03, B04.\\ &C00 \to B00, C01, C02, C03, C04.\\ &\quad\vdots\\ &H00 \to G00, H01, H02, H03, H04.\\ &I00 \to H00, I01, I02, I03, I04, I05.\end{aligned}$$

FI1: *Full independent.* This is set FLR1 with the links between subsets $A00, B00, C00, \ldots$ removed.

IE1: *Independent equivalent.*

$$A00 \rightarrow A01, A02, A03, A04, A05.$$
$$A04 \rightarrow A00, A01, A02, A03, A05.$$
$$B00 \rightarrow B01, B02, B03, B04, B05.$$
$$B04 \rightarrow B00, B01, B02, B03, B05.$$
$$C00 \rightarrow C01, C02, C03, C04, C05.$$
$$C04 \rightarrow C00, C01, C02, C03, C05.$$
$$D00 \rightarrow D01, D02, D03, D04, D05.$$
$$D04 \rightarrow D00, D01, D02, D03, D05.$$

Supersets were created from each of the basic set classes. These sets can be distinguished from the basic set classes by the suffixes 2, 3, and 4. For example, CD2 represents a clustered dependent superset created by the union of two unique CD1 sets. CD3 and CD4 represent supersets created by the union of 3 and 4 unique CD1 sets, respectively.

Composite sets were created from the union of independent equivalent set classes with sets of functional dependencies previously discussed in Section 3.

C1A: *Composite* 1*A*. Set IE1 and A00 → A06.

$$C10 \rightarrow T10.$$
$$T10\,H10 \rightarrow R10.$$
$$H10\,S10 \rightarrow R10, C10.$$
$$H10\,R10 \rightarrow C10.$$
$$C10\,S10 \rightarrow G10.$$

C1B: *Composite* 1*B*. Set C1A and $A11\,C11\,D11 \rightarrow B11$.

$$A11\,B11 \rightarrow C11.$$
$$B11\,C11 \rightarrow D11.$$
$$B11\,E11 \rightarrow C11.$$
$$C11\,G11 \rightarrow B11, D11.$$
$$C11\,E11 \rightarrow A11, G11.$$
$$C11 \rightarrow A11.$$
$$D11 \rightarrow E11, G11.$$

C1C: *Composite* 1*C*. Set C1B and $A12 \rightarrow B12$, $C12$, $D12$, $E12$, $F12$, $G12$, $H12$.

$$D12 \rightarrow B12, B12, H12.$$
$$H12 \rightarrow D12.$$
$$G12 \rightarrow K12, L12, M12, N12.$$

A second class of composite sets was created by using IE2 instead of IE1. The sets C2A. C2B, C2C are of this class.

APPENDIX B. BENCHMARKS FOR ALGORITHM 3.2.1

TABLE 1

Benchmarks for Clustered Dependent Sets

	CD1	CD2	CD3	CD4
Number of FDs	41	82	123	164
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.055	0.044	0.044	0.055
Step 2	6.551	30.593	71.183	130.426
Step 3	0.308	1.000	2.131	3.724
Step 4	0.439	2.285	6.075	12.435
Step 5	0.055	0.033	0.055	0.033
Total	7.481	33.955	79.488	146.673
Bernstein	4.833	21.311	49.048	88.705

TABLE 2

Benchmarks for Clustered Independent Sets

	CI1	CI2	CI3	CI4
Number of FDs	40	80	120	160
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.055	0.055	0.033	0.055
Step 2	5.053	21.201	48.469	87.046
Step 3	0.275	0.868	1.824	3.142
Step 4	0.439	2.252	6.042	12.292
Step 5	0.055	0.044	0.055	0.033
Total	5.877	24.420	56.452	102.568
Bernstein	3.460	13.951	31.518	56.079

TABLE 3

Benchmarks for Linear Reversed Sets

	LR1	LR2	LR3	LR4
Number of FDs	50	100	150	200
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.033	0.033	0.033	0.055
Step 2	1.505	5.954	13.248	23.398
Step 3	0.253	0.890	1.933	3.372
Step 4	0.055	0.055	0.044	0.055
Step 5	0.055	0.055	0.044	0.055
Total	1.900	7.008	15.302	26.935
Bernstein	1.044	4.064	9.118	16.258

TABLE 4

Benchmarks for Full Linear Reversed Sets

	FLR1	FLR2	FLR3	FLR4
Number of FDs	45	90	135	180
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.055	0.044	0.055	0.044
Step 2	18.345	102.810	249.395	455.837
Step 3	0.143	0.352	0.670	1.142
Step 4	0.044	0.044	0.044	0.055
Step 5	0.044	0.044	0.055	0.055
Total	18.631	103.293	250.219	457.134
Bernstein	16.423	86.233	208.387	394.310

TABLE 5

Benchmarks for Full Independent Sets

	FI1	FI2	FI3	FI4
Number of FDs	37	74	111	148
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.055	0.044	0.055	0.055
Step 2	2.461	10.073	22.629	39.382
Step 3	0.110	0.253	0.516	0.868
Step 4	0.055	0.044	0.033	0.044
Step 5	0.055	0.044	0.033	0.044
Total	2.735	10.458	23.266	40.370
Bernstein	1.922	7.635	17.082	30.923

TABLE 6

Benchmarks for Independent Equivalent Sets

	IE1	IE2	IE3	IE4
Number of FDs	40	80	120	160
Minimal covers	1	1	1	1
Time (sec):				
Step 1	0.055	0.044	0.033	0.044
Step 2	4.010	16.291	36.833	65.724
Step 3	0.198	0.604	1.252	2.175
Step 4	0.055	0.055	0.044	0.055
Step 5	0.055	0.055	0.044	0.055
Total	4.361	17.049	38.206	68.053
Bernstein	0.989	3.625	8.019	14.116

TABLE 7

Benchmarks for Composite Sets

	C1A	C1B	C1C	C2A	C2B	C2C
Number of FDs	47	58	73	87	98	113
Minimal covers	2	4	8	2	4	8
Time (sec):						
Step 1	0.417	1.813	2.142	0.615	2.746	3.131
Step 2	5.569	7.733	12.226	19.202	23.838	31.626
Step 3	0.373	0.747	1.088	0.824	1.736	2.252
Step 4	0.132	1.208	6.349	0.176	1.835	8.887
Step 5	0.044	0.044	0.055	0.044	0.044	0.055
Total	6.536	11.547	21.860	20.861	30.376	45.962
Bernstein	1.703	3.955	5.712	4.998	8.678	11.425

REFERENCES

1. C. Beeri and P. Bernstein, Computational problems related to the design of normal form relational schemas, *ACM Trans. Database Systems* 4(1):30–59 (Mar. 1979).
2. C. Beeri and M. Kifer, An integrated approach to logical design of relational database schemes, *ACM Trans. Database Systems* 11(2):134–158 (1986).
3. Philip Bernstein, Synthesizing third normal form relations from functional dependencies, *ACM Trans. Database Systems* 1(4):247–198 (Dec. 1976).
4. S. Ceri and G. Gottlob, Normalization of relations and Prolog, *Comm. ACM* 29(6):524–544 (June 1986).
5. E. Codd, A relational model of data for large shared data banks, *Comm. ACM*, 1970, pp. 377–387.
6. C. Date, *An Introduction to Database Systems*, Vol. 1, 4th ed., Addison-Wesley, Reading, Mass., 1986.
7. J. Diederich and J. Milton, New Methods and Fast Algorithms for Database Normalization, *ACM Trans. Database Systems* 13(3):339–365 (Sept. 1988).
8. R. Fagin, Multivalued dependencies and a new normal form for relational databases, *ACM Trans. Database Systems* 2(3):262–278 (Sept. 1977).
9. D. Maier, Minimum covers in the relational database model, *J. Assoc. Comput. Mach.* 27(4):664–674 (Oct. 1980).
10. D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville, Md., 1983.
11. J. D. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, Rockville, Md., 1982.
12. C. Zaniolo and M. Melkanoff, On the design of relational database schemata, *ACM Trans. Database Systems* (1):1–47 (Mar. 1981).
13. C. Zaniolo and M. Melkanoff, A formal approach to the definition and the design of conceptual schemata for database schemata, *ACM Trans. Database Systems* 7(1):24–59 (Mar. 1982).

Received 30 December 1987; revised 8 May 1988