INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University
Microfilms
International

300 N. ZEEB ROAD, ANN ARBOR, MI 48106
18 BEDFORD ROW, LONDON WC1R 4EJ, ENGLAND

Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/4803

8101522

POURNAGHSHBAND, HASSAN

A NEW APPROACH FOR CONSTRUCTING A RELATIONAL SCHEMA FROM A SET OF DATA DEPENDENCIES

The University of Oklahoma PH.D. 1980

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

THE UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A NEW APPROACH FOR CONSTRUCTING A RELATIONAL SCHEMA FROM A SET OF DATA DEPENDENCIES

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

DOCTOR OF PHILOSOPHY

BY

HASSAN POURNAGHSHBAND

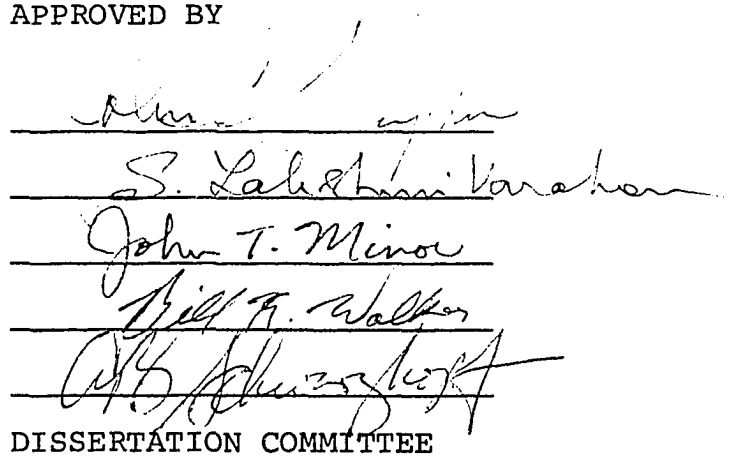
Norman, Oklahoma

1980

A NEW APPROACH FOR CONSTRUCTING A

RELATIONAL SCHEMA FROM A SET OF DATA DEPENDENCIES

APPROVED BY



DISSERTATION COMMITTEE

To my lovely mother, Haji Khanoom.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation and deep gratitude to Dr. John C. Thompson, for his keen interest, capable guidance, and invaluable encouragement during my graduate program and particularly throughout the planning and completion of this study.

I am also sincerely thankful to the members of my dissertation committee, Dr. S. Lakshmivarahan, Dr. John T. Minor, Dr. Albert B. Schwarzkopf, and Dr. Bill K. Walker, for their helpful and valuable suggestions.

It is also my pleasure to thank Ms. Betty Sudduth for her expert typing of this manuscript.

My special gratitude to my family, Haji Khanoom, Nahid and Ahmad, Roohangiz and Hossein, Krista and Reza, Pouri and Mehdi, Hayedeh and Javad, and Massoud, for their patience and continuing encouragement throughout my academic career.

Finally, my especial thanks go to someone who has made my life worthwhile.

ABSTRACT

It is the purpose of this work to investigate the Third and Fourth normal form relational schemas.

The relational model and the notion of relational data bases were first presented by Codd. Codd introduced First, Second, the Third normal forms, and presented an approach to the 3NF decomposition. He defined a normalized relation as one for which each of the underlying domains contains atomic values only, so that every value in the relation is in turn atomic.

In addition to Codd's decomposition approach, another approach to the logical design of relational data bases was the synthetic approach. Among several attempts to this approach, Bernstein's algorithm is efficient and has been proved to be correct.

A new normal form (4NF) was proposed by Fagin in 1977. Fagin presented a decomposition approach to the 4NF relations. His decomposition procedure leads to a family of 4NF relations which is not necessarily "optimal".

The purpose of this work is to present a new approach to constructing 4NF relations from functional dependencies

and multivalued dependencies. The objectives of the procedure are two-fold: To make the task as algorithmic as possible, and to produce an "optimal" 4NF family.

TABLE OF CONTENTS

		Page
ABSTRACT		iv
LIST OF ILLUSTRATIONS		ix
Chapter		
I.	INTRODUCTION	1
1.1	History	1
1.2	The Relational Model -- Definitions and Notations	6
1.3	Operations on Relations	9
1.4	Normalization	12
1.5	Data Dependencies	15
1.6	Normal Forms	18
II.	THE ALGEBRA OF FUNCTIONAL DEPENDENCIES AND MULTIVALUED DEPENDENCIES	22
2.1	Introduction	22
2.2	The Inference Rules for Data Dependencies	22
2.2.1	Armstrong's Axiomatization of Functional Dependencies	23
2.2.2	Inference Rules for Multivalued Dependencies	26
2.2.3	Inference Rules for Mixed Dependencies	30
2.3	The Closures of Dependencies	32
2.4	The Principles of Schema Design	36
2.5	Chapter Summary and Remarks	38
III.	CONSTRUCTING RELATION SCHEMES FROM DATA DEPENDENCIES	39
3.1	Introduction	39
3.2	General descriptions of the Approach	40

Chapter Page

3.3 A Simple Dependency-to-Relation Method 41
3.3.1 Proof of Representation 43
3.3.2 Proof of Normalization 44
3.4 The Dependency-to-Relation Method, 1st Improvement 46
3.4.1 Proof of Representation 46
3.5 The Dependency-to-Relation Method, 2nd Improvement 51
3.5.1 Proof of Representation 52
3.6 The Dependency-to-Relation Method, 3rd Improvement 57
3.6.1 Proof of Representation 58
3.6.2 Proof of Normalization 62
3.7 The Dependency-to-Relation Method, 4th Improvement 65
3.7.1 Proof of Representation 70
3.7.2 Proof of Normalization 71
3.8 The Dependency-to-Relation Method, Final Improvement 75
3.9 Chapter Summary and Remarks 84

IV. CONSTRUCTING CLOSURE II AND CLOSURE III . . . 85

4.1 Introduction 85
4.2 Description of the Closure II Algorithm. 85
4.2.1 Proof of Termination 89
4.2.2 Proof of Correctness 89
4.2.3 Output Analysis 92
4.2.4 Speed Analysis 95
4.3 Description of the Closure III Algorithm 95
4.3.1 Proof of Correctness 96
4.3.2 Proof of Termination 101
4.3.3 Output Analysis 101
4.3.4 Speed Analysis 102

Chapter Page

4.4 Description of the Dectection Algorithms 103
4.5 Chapter Summary and Remarks 103

V. ANALYSIS OF RELATIONS BY DEPENDENCY-TO-RELATION 106

5.1 Introduction 106
5.2 Properties of Relations Constructed by FNF Algorithms 106

5.2.1 Representation Principle 107
5.2.2 Separation Principle 113
5.2.3 Minimal Redundancy Principle . . . 117

5.2.3.1 Description of the Optimal Decomposition Algorithm . 129

5.3 Bernstein's Synthetic Approach 131
5.4 Fagin's Decomposition Approach 134
5.5 Chapter Summary and Remarks 137

VI. SUMMARY AND CONCLUSION 138

6.1 Summary 138
6.2 Analysis of the Approach 139
6.3 Suggestions for Further Work 141

BIBLIOGRAPHY 143

APPENDIX 146

LIST OF ILLUSTRATIONS

Figure		Page
1-1	An example of a relation	2
1-2a	An example of relational schema (consisting of three relational schemes). . .	8
1-2b	A snapshot of the schema of part "a".	8
1-3	The projection operation	
1-4	The join operation	11
1-5a	An example of a relation with undesirable properties	13
1-5b	A different normal form of the schema of part (a), without undesirable properties . .	13
1-6a	A third normal form schema with undesirable properties	16
1-6b	The fourth normal form of the schema of part "a", without undesirable properties . .	16
1-7	Transitive Dependency	20
3-1	Algorithm 3-1	42
3-2	Algorithm 3-2	47
3-3	Algorithm 3-3	53
3-4	Algorithm 3-4	59
3-5	Derivation of A $\rightarrow$ C from A,D $\rightarrow$ B,C and A $\rightarrow\!\!\!\!\rightarrow$ D	64
3-6	Algorithm 3-5	66
3-7	Algorithm 3-6	76
4-1	Algorithm 4-1	87
4-2	Algorithm 4-2	97

Figure		Page
4-3	Algorithm 4-3	104
5-1a	An instance of the schema R	109
5-1b	An instance of the schema S	109
5-2	An instance of the schema S'.	111
5-3	Algorithm 5-1	130
5-4	Algorithm 5-2	132

CHAPTER I

INTRODUCTION

1.1 History

One of the most significant contributions to data management technology in recent years has been the development of the relational point of view of a data base. In this section we will give a very informal discussion of the relational approach and its problems and will trace its historical developments.†

The concept of the relational model of data was first proposed in 1970 by Codd [10]. Conceptually, a relation can be viewed as a table, such as the one in Figure 1-1. Note that the table resembles the traditional file, with rows corresponding to records of the file and columns corresponding to fields of the records [13]. This tabular representation of data makes the computer accessible not only to the professional users, but also to the casual users with little or even no programming background. The conceptual simplicity of the relational model together with its mathematical elegance have attracted a large number of followers. This

† Formal definitions and technical discussions are given later.

An example of a relation

HOUSING

TENANT	COMPLEX	TYPE	RENT	APT#	MANAGER
Jack	Nieman	Fur.	100	17D	Smith
Mary	Parkview	Unfur.	110	212H	Brown
Phil	Kraettli	Fur.	200	305F	Rogers
Mark	Kraettli	Fur.	200	302A	Rogers
Mike	Kraettli	Fur.	200	208C	Rogers
John	Const.	Unfur.	175	126H	Rose
Tom	Const.	Unfur.	175	113F	Rose

Figure 1-1

dissertation deals mainly with the relational model, and specifically it considers the problem of "logical schema design", i.e. how relations should be designed to take advantage of the relational model's strengths.

Shortly after Codd introduced the relational model for data bases, he observed that certain relationships (functional dependencies) contained in a relation can cause data manipulation anomalies (i.e., adding new information, modifying existing information, or deleting old ones might be difficult and sometimes impossible). Therefore, he proposed a hierarchical set of constraints on relations. These constraints restrict relations to certain canonical forms, called Normal forms by Codd. The most desirable of these is Third normal form (3NF)[11]. Subsequently several researchers proposed various algorithms for constructing 3NF relations [8,15,29].

All of the above works are based on the axiomatization of functional dependencies developed by Armstrong [3]. Wang and Wedekind proposed an algorithm to synthesize the relational schema from a set of functional relationships [29]. However, their algorithm is not correct, in the sense that it could generate two relations with keys that are functionally equivalent.

Another approach was made by Bernstein [8]. His work was based on the approach given by Delobel and Casey [15]. Although the relational schema produced by Bernstein's

algorithm is in 3NF, it may contain properties that can cause data manipulation anomalies. This problem, which was discovered by Fagin [18] and independently by Zaniolo [31], stems from the fact that there may exist a special type of relationship between attributes which is not functional (a so-called multivalued dependency). Considering this new concept of relationship, Fagin defined a new normal form which he called fourth normal form (4NF). Then he proposed a decomposition approach for constructing 4NF relations [18]. His process begins by forming a single relation consisting of all attributes of the data base. Then this relation is broken down into two subrelations in a more desirable form (i.e., causing less data anomalies). This process continues for each subrelation not in 4NF, until all of them are in 4NF. The family of 4NF relations constructed in this way is not necessarily "optimal", and may contain redundant information. In addition, for a large data base, the original relation produced by Fagin's approach can be extremely large (it consists of all attributes of the data base), and thus makes the problem (i.e., breaking the relations) costly and time consuming.

The main objective of the new approach given in this thesis is also to construct 4NF relations. In this approach (in contrast to Fagin's method), the original and final relations are approximately of the same size. This is in fact the primary advantage of the approach. Briefly, the process

begins by constructing one relation for each relationship that exists between any pair of attributes (or sets of attributes). Then these relations are broken down to sub-relations until all of them are in 4NF. The relational schema constructed by this approach contains the theoretical minimal number of attributes and thus optimizes the use of computer memory.

The organization of the thesis is as follows: In this chapter, basic definitions and notations are given. Also a detailed discussion about normalization and normal forms are presented.

In Chapter II, the theoretical bases for designing a relational data model in general, and for our new approach in particular, are discussed.

The algorithm for constructing a relational schema in Fourth Normal Form is presented in Chapter III. The chapter starts with a simple algorithm which produces a schema not necessarily in 4NF. Then this simple algorithm is modified to eliminate undesirable properties of the schema as much as possible. This modification process is continued until we come up with an algorithm (which in this dissertation is called Fourth Normal Form Algorithm, or briefly, FNF) which produces a schema in 4NF. Theoretical issues and all proofs regarding the above algorithms are also described in Chapter III.

In Chapter IV, two algorithms are introduced for finding Closure II and Closure III (two sets that play significant

roles in our approach) for a given set of data dependencies. The feasibility of these algorithms and proofs for their correctness are also discussed in this chapter.

In Chapter V, properties of relations constructed by the FNF algorithm are examined and they are compared with relations constructed by other approaches. Also in this chapter a detailed discussion about "optimal" schemas is given.

Finally, in Chapter VI, an investigation of the practical and theoretical consequences of our approach is given. Also, several directions for further research are pointed out at this time.

The proofs of those problems related to this research that have been dealt with in previous work by others are given in the appendix.

1.2 The Relational Model -- Definitions and Notations

Conceptually, a relation can be viewed as a table in which each row corresponds to a distinct entity (or tuple) and each column to a distinct attribute. There exists a set of possible associated values, called the domain of attribute for each attribute.

Formally, a relation can be defined as follows: Given a collection of sets $D_1, D_2, \ldots, D_n$ (not necessarily distinct), a relation of R, defined over $D_1, D_2, \ldots, D_n$, is a subset of the cartesian product $D_1 \times D_2 \times \ldots \times D_n$. That is, R is a set of n-elements each of the form $(d_1, d_2, \ldots, d_n)$ where $d_i \in D_i$, for $1 \leq i \leq n$. Each element of the R is called a tuple of R. R is said to be a relation of degree n. An attribute is a

name assigned to a domain of a relation. While the domains of a relation need not be distinct, the attribute names assigned to them must be unique within the relation.

Suppose $A_1, A_2, \ldots, A_n$ are the names of the domains $D_1, D_2, \ldots, D_n$ of a relation R, then we use notation 1.2.1 for R.

$$R(A_1, A_2, \ldots, A_n) \tag{1.2.1}$$

The attribute set of R is defined as

$$U = \{A_1, A_2, \ldots, A_n\} \tag{1.2.2}$$

We will also use (1.2.3) to designate R on the set of attributes U.

$$R(U) \tag{1.2.3}$$

The structure of a relation is sometimes called the intention (scheme) and the contents of a relation is referred to as the extension. The contents of a relation may vary from time to time. That is, tuples may be modified, deleted from a relation, or/and inserted in a relation. The contents of a relation in a particular time is called its snapshot (or instance). Figure 1-2 indicates a schema (i.e., a collection of schemes) consisting of three relation schemes, $R_1(A,B)$, $R_2(C,D,E)$ and $R_3(B,D,F)$; and a snapshot of the schema.

Let R(U) be a relation on the set of attributes U, and let W be a subset of U, then W is a candidate key of R if it can uniquely identify the tuples of R and no proper subset of W has this property. A relation may

A Relational Schema and its Snapshot

a) An example of relational schema (consisting of three relational schemes)

$R_1(A,B) = \{(a_1,b_2),(d_2b_4),(d_3,b_1)\}$

$R_2(C,D,E) = \{(c_1,d_1,e_2),(c_2,d_3,e_1)(c_3,d_2,e_1),(a_1,d_3,e_3)\}$

$R_3(B,D,F) = \{(b_1,d_2,f_3),(b_2,d_2,f_2),(b_3,d_1,f_1)\}$

b) A snapshot of the schema of part "a"

R_1(A,	B)
a_1	b_2
a_2	b_4
a_3	b_1

R_2(C,	D,	E)
c_1	d_1	e_2
c_2	d_3	e_1
c_3	d_2	e_1
c_4	d_3	e_3

R_3(B,	D,	F)
b_1	d_2	f_3
b_2	d_2	f_2
b_3	d_1	f_1

Figure 1-2

have more than one candidate key. An attribute is said to be prime if it appears in any candidate key of the relation, and it is called a non-prime attribute otherwise.

1.3 Operations on Relations

Codd has defined [10] a number of operations on relations. In this dissertation two operations are of particular interest: Projection and Join.

Let R be a relation defined on the set of attributes $U = A_1, A_2, \ldots, A_n$. For any $W = A_1, A_2, \ldots, A_m$, that is $W \subseteq U$, the projection of R on W is defined as:

$$\Pi \downarrow R(W) = (a_1, a_2, \ldots, a_m) \ (a_1, a_2, \ldots, a_n) \in R \quad (1.3.1)$$

In other words, we can think of the projection of R onto W as the operation that takes the relation (instance) represented by R, then deletes all columns except those labeled by attributes in W, and finally identifies common tuples (See Figure 1-).

The join operator which is in some senses an inverse to the projection operator, in fact connects attributes of different relations together. The join (natural join) of a relation R(X,Y) with a relation P(Y,Z) on Y, is defined as:

$$R * P = \{(x,y,z) \mid (x,y \in R \text{ and } (y,z) \in P\}. \quad (1.3.2)$$

Figure 1- indicates an example of join operation.

The projection operation

Suppose an instance of relation R(A,B,C) is

R(A,	B,	C,	D,	E)
a_1	b_3	c_2	d_1	e_3
a_1	b_2	c_3	d_1	e_4
a_2	b_4	c_1	d_3	e_4
a_3	b_1	c_2	d_2	e_2
a_3	b_1	c_3	d_2	e_1

Then the projection of R on A, B and D is

R'(A,	B,	D)
a_1	b_3	d_1
a_1	b_2	d_1
a_2	b_4	d_3
a_3	b_1	d_2

Figure 1-3

The join operation

If instances of relations R(A,B,C) and P(C,D) are

R(A,	B,	C)
a_1	b_3	c_2
a_1	b_2	c_3
a_2	b_4	c_1
a_3	b_1	c_2
a_3	b_1	c_4

P(C,	D)
c_1	d_1
c_2	d_3
c_4	d_2

then the natural join of R with P (i.e., R * P) is

RP(A,	B,	C,	D)
a_1	b_3	c_2	d_3
a_2	b_4	c_1	d_1
a_3	b_1	c_2	d_3
a_3	b_1	c_4	d_2

Figure 1-4

1.4 Normalization

The notion of normalization in relational data base was first presented by Codd [11]. Codd observed that certain relations have structural properties that are undesirable for describing data bases. These undesirabilities stem from the fact that some attributes in a relation are related to each other in a certain way. For example, in relation TCM of Figure 1-5a, there is a relationship between COMPLEX and MANAGER. That is, given the complex, we can determine its manager. Note that in relation TCM the association between a complex and its manager is repeated for each TENANT. This repetition causes data manipulation anomalies. These anomalies can be categorized as follows:

(1) Update anomaly. Suppose the association between a complex and its manager is changed (i.e., the complex is no longer managed by the old manager, but a new one). Then we need to update all tuples and change the values of complex and manager for all tenants of the complex. On the other hand, if we update only one tuple, it will not be adequate for maintaining a consistant schema. This is known as update anomaly.

a) An example of a relation with undesirable properties. (The association between a complex and its manager is repeated for each tenant.)

TCM(TENANT,	COMPLEX,	MANAGER)
Jack	Nieman	Smith
Mary	Parkview	Brown
Phil	Kraettli	Rogers
Mark	Kraettli	Rogers
Mike	Kraettli	Rogers
John	Const.	Rose
Tom	Const.	Rose

b) A different normal form of the schema of part "a", without undesirable properties.

TC(TENANT,	COMPLEX)	CM(COMPLEX,	MANAGER)
Jack	Nieman	Nieman	Smith
Mary	Parkview	Parkview	Brown
Phil	Kraettli	Kraettli	Rogers
Mark	Kraettli	Const.	Rose
Mike	Kraettli		
John	Const.		
Tom	Const.		

Figure 1-5

(2) Insertion anomaly. If a new TENANT moves into a COMPLEX, and he/she happens to be the first tenant of the complex, then an association between the complex and manager will also be needed to insert the new tuple. This is called insertion anomaly.

(3) Deletion anomaly. If a TENANT moves out from a complex, and he/she happens to be the last tenant of the complex (i.e., no more tenants are residing in the complex), then by removing his tuple from the relation, the association between the complex and its manager also disappears from the relation. This is known as deletion anomaly.

Data manipulation anomalies discussed above will disappear if we convert the schema into a "better" normal form. (See Figure 1-5b).

For a relational schema, in fact, there are usually different sets of relation schemes that can represent all of the information needed. Thus, to avoid data manipulation anomalies attempts have been made to introduce schemas with no undesirable structural properties for describing data bases. These considerations led Codd to define a series of three normal forms [11], First Normal

Form, Second Normal Form, and Third Normal Form.

Later in 1977, Fagin [18] discovered that even by putting a schema into Third Normal Form, not all of the anomaly problems necessarily disappear. This led him to propose a new normal form, called Fourth Normal Form. An example of a Third Normal Form schema with undesirable properties is illustrated in Figure 1-6a. Figure 1-6b shows the Fourth Normal Form of the schema.

Data manipulation anomalies appear in this third normal form schema, because of the undesirable relationship that exists between attributes EMPLOYEE and CHILD in relation EDC.

A formal description of all normal forms is given in section 1.6.

1.5 Data Dependencies

The problems associated with constructing Fourth Normal Form relational schemas are tied to the fact that some attributes determine the values of other attributes. This can be formalized as the comcept of Functional Dependency (FD) and Multivalued Dependency (MVD).

The concept of functional dependency is defined as follows: if X and Y are distinct collections of attributes of some relation scheme R, then Y is said to be functionally dependent on X (or X functionally determines Y) if, at every point in time, each X value has no more than one Y

a) A Third Normal Form schema with undesirable properties. (The association between an employee and his/her department is repeated for each child.)

EDC(EMPLOYEE,	DEPARTMENT,	CHILD)
Johnson	Accounting	Henry
Johnson	Accounting	Bonnie
Johnson	Accounting	Ralph
Stewart	Personnel	Tim
Stewart	Personnel	Martha
Jones	Marketing	Joe

b) The Fourth Normal Form of the schema of part "a", without undesirable properties.

ED(EMPLOYEE,	DEPARTMENT)
Johnson	Accounting
Stewart	Personnel
Jones	Marketing

EC(EMPLOYEE,	CHILD)
Johnson	Henry
Johnson	Bonnie
Johnson	Ralph
Stewart	Tim
Stewart	Martha
Jones	Joe

Figure 1-6

value associated with it under the relation R. The notation used to denote that Y is functionally dependent on X in R is

$$R.X \longrightarrow R.Y \qquad (1.4.1)$$

and if no confusion exists, then R can be omitted.

$$X \longrightarrow Y \qquad (1.4.2)$$

The concept of Multivalued Dependencies has been proposed by Fagin in [18] as follows:

Let $R(X_1,\ldots,X_m,Y_1,\ldots,Y_n,Z_1,\ldots,Z_r)$ be a relation with $m + n + r$ attribute names. For notation convenience, we write X for $X_1,\ldots,X_m$; Y and Z are defined analogously. Whenever we write, say, $R(X,Y,Z)$, we assume automatically X, Y, and Z are pairwise disjoint as above. If $x_1,\ldots,x_m$ are entries that appear under columns $X_1,\ldots,X_m$, then we write x for $(x_1,\ldots,x_m)$; y and z are defined analogously. Define Y_{xz} to be $\{y:(x,y,z) \in R\}$. Of course Y_{xz} is nonempty if and only if x and z appear together in a tuple of R. Now, using the following notation to denote that X multidetermines Y in R

$$R.X \longrightarrow\!\!\!\!\!\rightarrow R.Y \qquad (1.4.3)$$

and when no confusion exists

$$X \longrightarrow\!\!\!\!\!\rightarrow Y \qquad (1.4.4)$$

the multivalued dependency $X \longrightarrow\!\!\!\!\!\rightarrow Y$ is said to hold for $R(X,Y,Z)$ if Y_{xz} depends only on x; that is, if $Y_{xz} = Y_{xz'}$, for each x,z,z' such that Y_{xz} and $Y_{xz'}$ are nonempty. As we can see the validity of MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ depends not only on

the values of attributes X and Y, but also on the value of Z, the complement of XY. This context dependency of MVDs makes the problem of schema design much more complicated than if only FDs were involved.

As an example of Multivalued dependencies see relation EDC of Figure 1-6, for which multivalued dependency EMPLOYEE $\longrightarrow\!\!\!\!\rightarrow$ CHILD holds.

1.6 Normal Forms

The concepts of functional dependence and multivalued dependence play significant roles in the theory which governs the decomposition of relations into subrelations in normal forms.

To show how a certain undesirable dependency creates problems discussed earlier, we will discuss the concepts of partial dependencies (and fully dependencies) and transitive dependencies mentioned by Codd [10,11]. We will also discuss the Fagin's [18] notion of nontrivial multivalued dependency.

We say that, Y is fully dependent on X in relation R, if

(1) X and Y are two distinct subcollections of attributes of relation R,

(2) R.X $\longrightarrow$ R.Y, and

(3) Y is not functionally dependent on any proper subset of X.

If condition (3) is not satisfied, then we say, Y is partially dependent on X in relation R.

Partial dependencies can create data manipulation anomalies and thus, have to be removed from the schema. This led Codd to further normalize the first normal form relations to get the second normal form [11].

A relation R is said to be in First Normal Form (1NF), if and only if all underlying domains contain atomic values only [13].

A relation R is said to be in Second Normal Form if:

(1) it is in first normal form, and

(2) every non-prime attribute of R is fully dependent on each candidate key of R.

To define third normal form relations, we need to know the concept of "transitive dependencies".

Given a relation R, further suppose that X, Y, and Z are three distinct collections of attributes of R, and if the following conditions are true

(1) $R.X \longrightarrow R.Y$

(2) $R.Y \not\longrightarrow R.X$

(3) $R.Y \longrightarrow R.Z$

then it follows that

$R.X \longrightarrow R.Z$ and

$R.Z \not\longrightarrow R.X$

Here, Z is said to be transitively dependent on X under the relation R. This concept is depicted in Figure 1-7.

Transitive Dependency

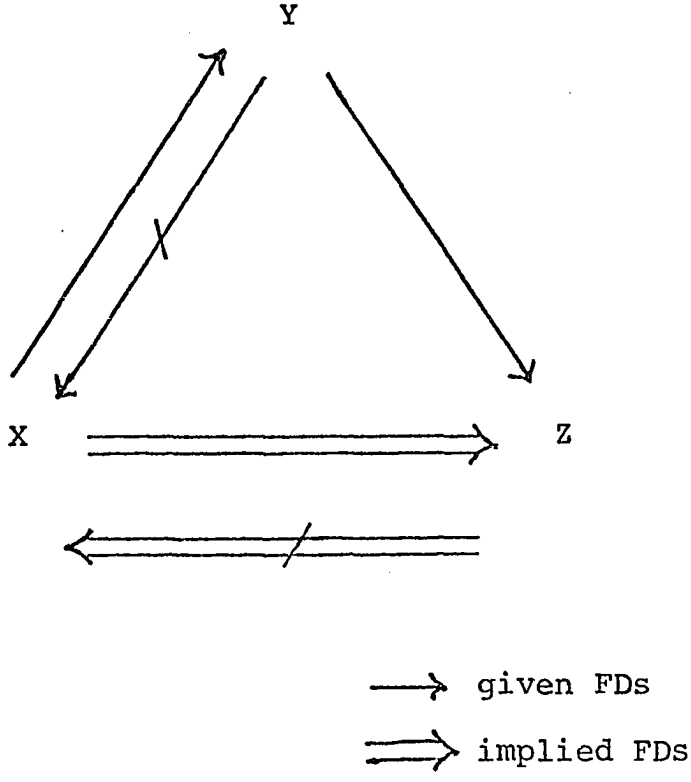


Figure 1-7

A relation is said to be in Third Normal Form (3NF) if,

(1) it is in second normal form, and

(2) every non-prime attribute of R is non-transitively dependent on each candidate key of R.

The concept of "trivial multivalued dependencies" proposed by Fagin [18], is needed in describing the fourth normal form relations.

Given relation R(U) where U = {X,Y}, then the multivalued dependencies $X \twoheadrightarrow \emptyset$ and $X \twoheadrightarrow Y$ necessarily hold for R. These are called Trivial Multivalued Dependencies (TMD).

A relation R is said to be in Fourth normal Form (4NF), if whenever a nontrivial multivalued dependency $X \twoheadrightarrow Y$ holds for R, then so does the functional dependency $X \rightarrow A$ for every attribute A of R. An example of a schema in 4NF is given in Figure 1-6.

CHAPTER II

THE ALGEBRA OF FUNCTIONAL DEPENDENCIES AND MULTIVALUED DEPENDENCIES

2.1 Introduction

The theoretical bases for designing a relational data model in general, and for our new design approach in particular, are discussed in this chapter. The inference rules for functional dependencies and multivalued dependencies are given in section 2.2, and it is proved that these rules are complete in the sense that a data dependency g is a consequence of a set of dependencies G if and only if g can be derived from G by a sequence of applications of the rules.

In section 2.3 different closures of dependencies are defined, and it is shown that two of these closures are of particular importance for this work.

Finally, the three principles that should be considered in designing a relational schema are presented in section 2.4. These principles are formally discussed in Chapter V.

2.2 The Inference Rules for Data Dependencies

When the problem of schema design is of concern,

it is important to know whether a dependency is implied by some other dependencies or not. Formally, this means that a dependency g is a consequence of a set of dependencies G if for all relation schemes R, g holds in R if all dependencies of G hold in R. In previous work, researchers have proposed and investigated the complete sets of inference rules for functional and multivalued dependencies [3,6,7,18, et al.]. Detailed discussions for these inference rules are given in the following subsections:

2.2.1 Armstrong's Axiomatization of Functional Dependencies

Axiomatization of functional dependencies was studied by Armstrong [3]. In [3] he has presented a set of axioms governing sets of functional dependencies. It is proved [3,7,19] that this set of axioms is complete for the family of FDs in the sense that, for a family of FDs, for each set F of FDs from the family, the FDs that are implied by F are exactly those dependencies that can be inferred from it using these inference rules. In other words, we say a set of axioms is complete for a family of FDs if and only if, for each set F of FDs over a set of attributes U, if F^+ is the set of FDs that follow logically from F, then every relation over U that satisfies F, also satisfies the functional dependencies in F^+. For any formal system, the completeness of the

set of inference rules is an important concept for the system. For the family of functional dependencies, as it is mentioned in [7], only if a complete set of axioms is used, the data base designer can be assured that he has complete knowledge of all FDs that hold in the data base. This is in fact the basic reason that the completeness of Armstrong's axioms has been an important basis for research in this area (including the present research).

The complete set of axioms for the family of functional dependencies is given below. In the rules, X, Y, Z and W are arbitrary subsets of U, where U is the set of all attributes. We write X for the set X, and XY for the union of two arbitrary sets.

FD Rules:

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

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

FD3 (Transitivity): If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Other Useful Rules:

FD4 (Pseudo-Transitivity): If $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$.

FD5 (Union): If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.

FD6 (Decomposition): If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

If α and β are attributes of a relation R, then by applying the Axiom FD1 to $X = \{\alpha,\beta\}$ we get $\alpha\beta \rightarrow \alpha\beta$, $\alpha\beta \rightarrow \alpha$, $\alpha\beta \rightarrow \beta$, $\alpha \rightarrow \alpha$, and $\beta \rightarrow \beta$.

Axiom FD2 means that, knowing f: $X \rightarrow Y$, we can construct another functional dependency, say g: $X,\delta \rightarrow Y$, where the attributes appearing on the left side of g consists of X plus some other extraneous attributes δ, whose values have no effect on the value of Y selected by g.

For Axiom FD3, assume one is given the dependencies f: $X \rightarrow Y$ and g: $Y \rightarrow Z$. The axiom claims that there is a dependency h: $X \rightarrow Z$. Symbolically, h(X,Z) is defined to be g(f(X),Z).

Notice that, in the above set of axioms, the Axioms FD1-FD3 are sufficient, and the other three axioms, that is FD4-FD6 are implied by the first three axioms. As an example, Axiom FD4 can be derived from the Axioms FD1-FD3 as follows: As our assumption we have f_1: $X \rightarrow Y$ and f_2: $YW \rightarrow Z$. Now, from f_1 and Axiom FD2 we get f_3: $XW \rightarrow YW$. By applying Axiom FD3 to f_2 and f_3 we can derive an FD $XW \rightarrow Z$, completing the claim. Similarly, it is easy to show that the other two axioms, FD5 and FD6, can also be derived from the first three axioms.

Let F be a set of functional dependencies over a set of attributes U. Then the <u>closure</u> of F, denoted by F^+, is defined to be the set of all functional dependencies that can be obtained by successive application of Axioms FD1,

FD2, and FD3 on the set F, over the set of attributes U. Notice that by Armstrong's theory, if F is a given set of FDs for a relation R, then each FD in F^+ also exsits in R.

A functional dependency $f_i \in F$ is said to be redundant in F if $F^+ = (F - f_i)^+$. Also, H is a nonredundant covering of a given set of functional dependencies F, if $F^+ = H^+$ and H contains no redundant FDs. As we will see later, the concepts of closures and covering are important in constructing the relational schemas from dependencies. Letting F^+ be the general closure of a given set of dependencies, we will present in Section 2.3 two special closures of a set of functional and multivalued dependencies. These two closures are of particular importance in constructing a relational schema from a set of data dependencies using the new design approach discussed in this research.

2.2.2 Inference Rules for Multivalued Dependencies

A set of inference rules for multivalued dependencies is presented in [7], and it is proved that the given set is complete for the family of MVDs. Similar to FDs, we say that a set of inference rules is complete for the family of MVDs, if for each set M of MVDs from the family, the MVDs that are implied by M are exactly those dependencies that can be inferred from it using these inference rules. The complete set of rules for multivalued

dependencies is given below. In the rules, X, Y, Z, and W are arbitrary sets of attributes. We use XY for the union of two arbitrary sets of attributes X and Y.

MVD Rules:

MVD0 (Complementation): If $U = XYZ$ and $Y \cap Z \subseteq X$, then $X \twoheadrightarrow Y$ if and only if $X \twoheadrightarrow Z$.

MVD1 (Reflexivity): If $Y \subseteq X$ then, $X \twoheadrightarrow Y$.

MVD2 (Augmentation): If $Z \subseteq W$ and $X \twoheadrightarrow Y$ then, $XW \twoheadrightarrow YZ$.

MVD3 (Transitivity): If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$ then, $X \twoheadrightarrow Z - Y$.

other useful rules:

MVD4 (Pseudo-Transitivity): If $X \twoheadrightarrow Y$ and $YW \twoheadrightarrow Z$ then, $XW \twoheadrightarrow Z - YW$.

MVD5 (Union): If $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$ then, $X \twoheadrightarrow YZ$.

MVD6 (Decomposition): If $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$ then, $X \twoheadrightarrow Y \cap Z$, $X \twoheadrightarrow Y - Z$, and $X \twoheadrightarrow Z - Y$.

It is interesting to note that for each FD rule there exists one MVD rule corresponding to it. But notice, that there is no FD rule corresponding to the complementation rule of MVDs. In fact, as it was mentioned in Chapter I, multivalued dependencies are sensitive to context, while functional dependencies are context independent. In other words, to know if the MVD

$X \twoheadrightarrow Y$ holds in a relation R(U), where U = XYZ, we also need to know the values of the attributes in the set U - XY, that is, Z. As is discussed in [18], an MVD $X \twoheadrightarrow Y$ may not hold in one relation, but in one of its projections. This is not true for the case of FDs, because if an FD $X \rightarrow Y$ holds in a relation R(U), it also holds in any other relation R(U') as long as $X \bigcup Y \subseteq U'$.

As for FD rules, the inference rules MVD0-MVD3 are sufficient for multivalued dependencies, and the other useful rules indicated above (i.e., MVD4-MVD6), can be derived by rules MVD0-MVD3. This claim can be formalized as follows [24]:

Proposition 2.1 The Decomposition Rule (i.e., MVD6) can be derived from MVD1-MVD3.

Proof: Suppose we have m_1: $X \twoheadrightarrow Y$ and m_2: $X \twoheadrightarrow Z$, then we need to prove that (i) $X \twoheadrightarrow Y \cap Z$ and (ii) $X \twoheadrightarrow Z - Y$. From MVD m_1 and Axiom MVD2 we get m_3: $X \twoheadrightarrow XY$, and from m_2 and Axiom MVD2 we get m_4: $XY \rightarrow Z$. Now by applying Axiom MVD3 to m_3 and m_4 we can derive m_5: $X \twoheadrightarrow Z - XY$. We can also get m_6: $X(Z - XY) \twoheadrightarrow Z$ and m_7: $X \twoheadrightarrow X(Z - XY)$ by applying Axiom MVD2 to m_2 and m_5 respectively. From MVDs m_6 and m_7, and Axiom MVD3 we can get m_8: $X \twoheadrightarrow Z - X(Z-XY)$. Now, it is easy to get m_9: $X \twoheadrightarrow Y \cap Z - X$ from m_8 (by boolean manipulations) which in turn can result to m_{10}: $X(X \cap Y \cap Z) \twoheadrightarrow (Y \cap Z-X)(X \cap Y \cap Z)$. This (i.e., m_{10}) obviously leads to m_{11}: $X \twoheadrightarrow Y \cap Z$,

completing the claim for part (i). For part (ii), we may apply Axiom MVD2 to m_5 to get m_{12}: $Z((X-Y) \cap Z) \twoheadrightarrow (X-XY)((X-Y) \cap Z)$ which in turn can lead us to m_{13}: $X \twoheadrightarrow Z - Y$ (boolean manipulations), thus completing the proof. #

As we saw in Proposition 2.1, the Decomposition Rule can be derived without using the Complementation Rule MVD0. In fact, in [7] it was concluded that neither Rule MVD5 nor Rule MVD6 can be derived without using the complementation rule. This claim has been rejected in [24] for the Rule MVD6, because as we saw in Proposition 2.1 the Rule MVD6 is obtainable without using the Rule MVD0. But notice that, the Union Rule can be derived only if the Complementation Rule is also involved in the derivation process [7,24]. This claim is formalized in the following proposition [24].

Proposition 2.2 The Union Rule (i.e., MVD5) cannot be derived from Rules MVD1-MVD3.

Proof: The proof is given in the Appendix.

The significance of the complementation rule is more formally stated by Mendelzon [24] and others [3,18]. Indeed, it has been proved that the complementation rule does not follow from MVD1-MVD6.

In [24], Mendelzon has also proved that there are only two minimal complete subsets of the MVD Rules, the sets {MVD0, MVD1,MVD3} and {MVD0,MVD1,MVD4}. These are important results that can be of particular interest in

designing a relational schema. We will return to this problem in Chapter III.

Recall that a complete set of inference rules for functional dependencies was given in section 2.2.1, and a complete set of inference rules for multivalued dependencies was given in this seciton. Now, it is important to notice that, although FD rules are sufficient when there are only FDs, and MVD rules are sufficient when when there are only MVDs, the combination of FD rules and MVD rules is not sufficient for the case that both kind of dependencies, functional and multivalued, are involved. In other words, when we have a set of functional and multivalued dependencies, not all of the obtainable dependencies from this set can be inferred by the applications of FD rules and MVD rules. Indeed, there are two additional rules that can be applied only when both kind of dependencies exists in the set. These rules are given in the following subsection.

2.2.3 Inference Rules for Mixed Dependencies

In subsections 2.2.1 and 2.2.2 we dealt with the inference rules for FDs and MVDs respectively. In fact, in each subsection we concentrated on one type of dependency. That is, we wanted to know what additional FDs (or MVDs) can be implied by a set of FDs (or MVDs). Now, suppose $G = F \bigcup M$ is a set of data dependencies, where F is a set of FDs and M is a set of MVDs. Also, suppose

that F' is a set of FDs inferred from F by applying FD rules, and M' is a set of MVDs inferred from M by the applications of MVD rules. Now the question is, does the set G' = F' $\bigcup$ M' contain all dependencies implied by G? If not, what are the additional rules that can be used to deduce them?

In [7] three rules are introduced for mixed dependencies. They are given below. In the rules X, Y, Z, and Z' arbitrary sets.

FD-MVD Rules:

FD-MVD1: If $X \rightarrow Y$ then $X \twoheadrightarrow Y$.

FD-MVD2: If $X \twoheadrightarrow Z$ and $Y \rightarrow Z'$ where $Z' \subseteq Z$ and $Y \cap Z = \emptyset$, then $X \rightarrow Z'$.

additional useful rule:

FD-MVD3: If $X \twoheadrightarrow Y$ and $XY \rightarrow Z$, then $X \rightarrow Z - Y$.

Note that, the first two rules (FD-MVD1 and FD-MVD2) combined with the FD rules and the MVD rules are sufficient for mixed dependencies, and the Rule FD-MVD3 can be derived from them. The complete proofs concerning these rules are given in the appendix. Briefly, the Rule FD-MVD1 simply follows from the definitions of functional and multivalued dependencies, and it means that an FD is also an MVD. Indeed, as it was mentioned in Chapter I, functional dependencies are special cases of multivalued dependencies (notice that the converse of this concept is not true).

As we saw, the set of rules introduced in subsections 2.2.1, 2.2.2, and 2.2.3 are complete for the family of functional and multivalued dependencies. The proof of this completeness which originates from [7] is given in the appendix.

2.3 The Closures of Dependencies

If $G = F \bigcup M$ is a set of data dependencies, where F is a set of FDs and M is a set of MVDs, then the closure of G denoted by G^+ is the set of FDs and MVDs that can be deduced from $F \bigcup M$ by repeated applications of FD rules, MVD rules, and mixed rules.

Recall from the previous section that the set of rules {FD1,FD2,FD3,MVD0,MVD1,MVD2,MVD3,FD-MVD1,FD-MVD2} is sufficient, and thus the axioms of this set are those that are crucial in designing a relational schema. In fact, as we saw before, the other axioms are implied by those given in the above set, and need not be explicitly considered in the design process. In addition, we claim that Axioms FD1 and MVD1 are not needed either for this work. This is simply because FD $X \rightarrow X$ (or MVD $X \rightarrow\!\!\!\!\rightarrow X$) means, having a set of attributes X, we can determine (or multidetermine) a set of attributes X, and clearly, having a set X at our disposal we do not have to determine it. We also claim that Axioms FD2 and MVD2 are not needed either. In fact, if a set of attributes X

determines (or multidetermines) Y, then our method realizes that any set which contains X also determines (or multidetermines) Y.

Axioms MVD0 and FD-MVD1 need not be considered (explicitly) either, because as we will see in Chapter III, these rules are implicitly considered in our schema design.

As will be discussed in more detail in Chapter III, the above concepts lead to two special closures that are of particular interest for our new design approach. We formally define these closures as follows:

Definition 2.1 If $G = F \bigcup M$ is a set of mixed dependencies (FDs and MVDs), the closure II of G, denoted by G^{II}, is defined to be the set of all dependencies that can be obtained by successive application of axioms FD4 and MVD4.

An efficient closure II Algorithm which computes the closure II of a given set of FDs and MVDs is given in Chapter IV. The feasibility of constructing the closure II of a set for the real world applications is also discussed in that chapter.

Definition 2.2 Let $G = F \bigcup M$ be a set of mixed dependencies (FDs and MVDs), then the closure III of G, denoted by G^{III}, is defined to be the set of all dependencies that can be obtained by successive application of axioms FD4, MVD4, and FD-MVD2.

In chapter IV, an efficient algorithm which computes

the closure III of a given set of functional and multi-valued dependencies is given, and its feasibility is discussed in detail.

In all the published work of relational data base design [4,8, et al.], the computation of the closure of a set of dependencies has been avoided. This is mainly because the closure of a set of dependencies can be extremely large even if the original set is small. In particular it is the Reflexivity and Augmentation Rules, which produce a large number of dependencies from a given set. On the other hand, the closure II and closure III of a set of dependencies will be fairly small, and therefore we need much less time to compute them than if we were to compute the general closure.

Here, an example is given to clarify these concepts, and a formal discussion with all necessary proofs is given in Chapter IV.

Example 2.1: Given G, the following set of FDs and MVDs:

f_1: $A \rightarrow B$

f_2: $B \rightarrow D$

m_1: $C \twoheadrightarrow D$

we can construct G^+, the closure of G as follows:
Applying reflexivity rules we get

f_3: $A \rightarrow A$

f_4: $B \rightarrow B$

f_5: $C \rightarrow C$

f_6: $D \rightarrow D$

m_2: $A \twoheadrightarrow A$

m_3: $B \twoheadrightarrow B$

m_4: $C \twoheadrightarrow C$

m_5: $D \twoheadrightarrow D$

applying augmentation rules we get

f_7: $A,C \rightarrow B$

f_8: $A,D \rightarrow B$

f_9: $B,A \rightarrow D$

f_{10}: $B,C \rightarrow D$

m_6: $C,A \twoheadrightarrow D$

m_7: $C,B \twoheadrightarrow D$

applying transitivity rules we get

f_{11}: $A \rightarrow D$

applying complementation rule we get

m_8: $C \twoheadrightarrow A,B$

applying Axiom MVD1 we get

m_9: $A \twoheadrightarrow B$

m_{10}: $B \twoheadrightarrow D$

and finally applying Axiom MVD2 we get

f_{12}: $C \rightarrow D$

Now, we need to get the union of this new set. That is, we have to remove those dependencies that are repeated in the set. This is actually what makes the problem

costly and time consuming.

The same process should be applied to this new set to get more dependencies (if any). We continue this process until no more newer set is derivable.

As is indicated in the above Example, constructing the closure of a set of dependencies is very costly and time consuming (a formal treatment to this problem is given in Chapter IV. On the other hand, if we want to construct the closure III (and/or closure II), then we need only to apply the transitivity rules and FD-MVD2. If we do so, we get $G^{III} = \{A \rightarrow D, C \rightarrow D\}$, and $G^{II} = \{A \rightarrow D\}$. By comparing G^{+} with G^{III} and G^{II}, for the same set of dependencies, we can observe how much faster an algorithm can construct the closure II (and/or closure III) of a set of dependencies, than if it was to construct the general closure of the set.

2.4 The Principles of Schema Design

When constructing a relational schema from a set of data dependencies, there are principles that have to be considered. In other words, since the intention is to construct schemas with no structural properties that are undesirable for describing data bases (these undesirable properties were discussed in Section 1.4), we should consider these principles when using the dependencies for our schema design. Therefore, we say, a schema is a desirable

one if it satisfies these principles.

The three principles of the schema design that are of interest for this work are proposed in [6]. These three principles are representation, separation, and nonredundancy.

Roughly speaking, representation means that all information of interest should somehow be represented in the schema. On the other hand, if any information is lost during the transformation of dependencies (i.e., the process of schema design), then the constructed schema violates the representation principle, and thus it can no longer be a desirable schema.

A formal discussion is given in Chapter V. We will give our own definitions of these principles, and will prove that the schema constructed by our approach satisfies them.

By separation, it is meant that in designing a relational schema, attempts should be made to represent the basic information separately. This is, in fact, the motivation behind the normalization process [13]. In Chapter V, we will formally discuss this issue, and will prove that the schema constructed by our algorithm is as normalized as possible, that is, it is in fourth normal form.

The nonredundancy principle (sometimes called minimal redundancy) requires that the representation be

nonredundant, that is, while the schema must represent all of the information of interest, but it should not contain any redundant information. Specifically speaking, dependencies that can be derived from other explicitly represented dependencies, need not be explicitly represented. In addition, attributes should be repeated in as few relations as possible. In Chapter V, heuristics are employed for this issue, and it is formally shown how a design approach can lead to an "optimal" schema.

2.5 Chapter Summary and Remarks

The theoretical bases for designing a relational schema have been discussed in details. A set of axioms (FD rules) for the family of functional dependencies has been given, and it is shown that these axioms are complete for this family. Also, a complete set of rules (MVD rules) has been presented in this chapter for the family of MVDs. It has been stated that the combination of FD rules and MVD rules is not sufficient for the family of functional and multivalued dependencies, and thus additional rules (FD-MVD rules) have been given to complete the set of rules for FDs and MVDs.

Furthermore, the three design principles, i.e., representation, separation, and nonredundancy have been introduced.

CHAPTER III

CONSTRUCTING RELATION SCHEMES FROM DATA DEPENDENCIES

3.1 Introduction

A design approach for constructing a relational schema from a set of functional and multivalued dependencies is examined in this chapter. Since the intention is to propose an approach satisfying the three design principles discussed in Chapter II, then it is shown in this chapter that the schema constructed by this method:

(i) "represents" all information of interest in the schema,

(ii) does not represent any "redundant" information and

(iii) is as "normalized" as possible, that is, it is in fourth normal form.

The proofs concerning these principles, and the proofs of the other related problems are also given in this chapter (except the proofs for the final algorithm -- these proofs will be given in Chapter V).

It is shown, that although the constructed schema has no structural properties that are undesirable for describing

our data base, but it is not necessarily "optimal". This issue will be examined in details in Chapter V. We will propose the heuristics that can be employed for designing an algorithm which can construct an optimal schema.

3.2. General Descriptions of the Approach

We start our work with designing a simple algorithm and show several undesirable properties of the schema constructed by this method. That is, we will prove and also show by examples that some of the design principles are not satisfied, and thus may cause data manipulation anomalies. Then we will modify the algorithm to eliminate those undesirable properties as much as possible. We will continue this process of modification until we come up with an algorithm which produces a "good" schema. This "good" schema will not necessarily be "optimal", and thus we will further modify the algorithm (in Chapter V) to construct a schema which is both, "good" and "optimal".

Following each algorithm one or more examples are given to clarify the concepts considered (and/or missed) by the algorithm.

Also, we will put into account some basic semantics properties that attributes and relationships among them (FDs and MVDs) apparently have in the real world.

As a real world example we will consider a "university housing" data base, which is used as a test case throughout

the iterative refinement of the algorithm.

3.3. A Simple Dependency-to-Relation Method

The problem of the schema design is indeed: given a set of dependencies (FDs and MVDs designated by the data base administrator), how to construct a relational schema. For this method, we present an algorithm which gets as input a set of dependencies, and generates as output a set of relation schemes with designated keys (Figure 3-1).

The algorithm simply constructs one relation scheme for each explicitly given dependency (and not for dependencies that can be inferred from the given set), and designates the keys as follows: (i) if the corresponding dependency is an FD, then the key of the relation scheme would be the attributes appearing on the left side of the dependency, and (ii) if the corresponding dependency is an MVD, then the key of the scheme would be all of the attributes appearing in the dependency.

As we will see later, the schema constructed by this algorithm has some structural properties that are not desirable for describing the data base. As we mentioned before, these undesirable properties will be eliminated by further modification of the algorithm.

Note that, for all of the algorithm presented in this chapter, we assume without loss of generality that there are no two FDs f_i and f_j with identical left sides. In fact,

Algorithm 3-1

A Simple Algorithm to Construct a Schema From a Set of Dependencies

INPUT: A set F of m FDs; and a set M of n MVDs.

OUTPUT: A set R of m + n relation schemes (in 1NF or better).

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=1}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme;

end;

$$R'' = \bigcup_{i=m+1}^{m+n} R_i;$$

<<put all schemes together>>

$R = R' \cup R''$;

end algorithm;

Figure 3-1.

if there are any, we can simply combine them, using Axiom FD5.

3.3.1. Proof of representation

A schema embodies a set of dependencies (FDs and MVDs) if each dependency is embodied in at least one relation of the schema. An FD, g: $X \longrightarrow Y$ is embodied in a relation $R(A_1,A_2,\ldots,A_n)$ if [8]:

(i) $X \subseteq \{A_1,A_2,\ldots,A_n\}$,

(ii) $Y \subseteq \{A_1,A_2,\ldots,A_n\}$, and

(iii) $x \in DOM(X)$ and $y \in DOM(Y)$ implies that $g(x) = y$ if and only if the tuple $\{x,y\}$ is in the projection of R on X and Y.

As the reader will notice, it is easy to generalize the idea to MVDs.

Theorem 3.1. Let R be a relational schema constructed from a set of dependencies G = FUM, using the algorithm 3-1, then R represents the same information as G (i.e., embodies F and M).

Proof: The algorithm constructs one relation (scheme) for each $f_i \in F$ and one relation for each $m_i \in M$. Thus we need only to prove that each of those relations embodies its corresponding dependency. Assume that the relation scheme $R_i(X,Y)$ is constructed by dependency g_i: $X \longrightarrow Y$ (or $X \longrightarrow\!\!\!\!\!\rightarrow Y$), then obviously $X \subseteq \{X,Y\}$, $Y \subseteq \{X,Y\}$. And since

the projection of $R_i(X,Y)$ on X and Y is the R_i itself, thus $x \in DOM(X)$ and $y \in DOM(Y)$ implies that $g_i(x) = y$ if and only if the tuple $\{x,y\}$ is in R_i, completing the proof. #

3.3.2. Proof of Normalization

The schema constructed by the simple algorithm, clearly is not in the desirable normal form (i.e., 4NF), but as it is formally stated below, it is in some degree of normal forms.

Theorem 3.2. Let R be a relational schema constructed from a set of dependencies, using the Algorithm 3-1, then R is in first normal form.

Proof: The proof follows directly from the definition of first normal form. #

As we saw in the previous subsections, our simple algorithm satisfies the representation principle, and can only guarantee a schema of 1NF. We will show by some examples that the nonredundancy principle and 4NF schema cannot be guaranteed by this algorithm.

Example 3.1: Suppose the following data dependencies are given by the data base administrator:

f_1: $A \longrightarrow B,C$

m_1: $D,E \twoheadrightarrow F$

m_2: $D \twoheadrightarrow E,F$

then, the algorithm constructs one relation scheme for each FD or MVD as follows: (underscored attributes are keys)

$R_1(\underline{A},B,C)$

R_2 (D,E,F)

R_3 (D,E,F)

and after removing relation R_3 (or R_2) from the schema, we get

R_1 (A,B,C)

R_2 (D,E,F)

Although the algorithm works well for the cases like what we had in the above example, but it may not be appropriate for other situations. The following example clarifies this concept.

Example 3.2: In our UNIVERSITY HOUSING data base, if each COMPLEX is managed by only one MANAGER, and if each MANAGER can manage a set of COMPLEXes, then we may have the following dependencies and their corresponding relation schemes:

f_1: COMPLEX $\longrightarrow$ MANAGER R_1 (COMPLEX, MANAGER)

m_1: MANAGER $\longrightarrow\!\!\!\!\rightarrow$ COMPLEX R_2 (MANAGER, COMPLEX)

As we can see, R_1 and R_2 bear the same information (in the sense of attributes), and R_1 represents not only what it must represent, but also what is represented by R_2 (in the sense that in R_1 not all of the attributes make the key of the relation). Thus, R_2 is redundant and must be removed from the schema.

We can eliminate the above problem, and all problems of this type (but not necessarily the violation of nonredundancy principle), simply by searching for relation schemes

with the same set of attributes and "stronger" relationship/s among them (recall that an FD is an stronger dependency than an MVD). Formal discussion for this issue is given in Chapter V. Here we will solve the problem by modifying our simple algorithm. This modification is considered in PART-2 of the following algorithm (Figure 3-2). Notice that, this part searches only the set of schemes R' constructed from FDs. It does not consider the set R" constructed from MVDs because, it is known from PART-1 of the algorithm, that there are not any pair of schemes in the set R" that have the same set of attributes.

3.4. The Dependency-to-Relation Method, 1st Improvement

As it was discussed in the previous subsection, the simple algorithm (Algorithm 3-1) constructs a schema which may have undesirable properties. The algorithm presented in this section (Algorithm 3-2) eliminates some of these undesirable properties, but not all of them. Thus the algorithm will be further modified in the subsequent subsections.

3.4.1. Proof of Representation

Theorem 3.3. Let R be a relational schema constructed from a set of dependencies G = FUM, using the Algorithm 3-2, then R represents the same information as G.

Proof: The algorithm first constructs one relation scheme for each dependency, and then removes a relation scheme R_i

Algorithm 3-2

An Algorithm to Construct a Schema From a Set of Dependencies; 1st Improvement

INPUT: A set F of m FDs; and a set M of n MVDs.

OUTPUT: A set R of relation schemes (in 1NF or better).

Part-1: <<this part constructs one relation scheme for each dependency>>

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=1}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme;

end;

$$R'' = \bigcup_{i=m+1}^{m+n} R_i;$$

Figure 3-2.

Figure 3-2 (continued)

```
<<put all relation schemes together>>
  R = R' U R";
end PART-1;
PART-2:  <<this part removes a relation R_i if it has the
           same set of attributes as R_j, and the relation-
           ships of its attributes are not as strong as R_j>>
   do for each R_i ε R  for 1 ≤ i ≤ r - 1; <<r is cardinality
            of R>>
      do for each R_j ε R  for i + 1 ≤ j ≤ r;
        if U_Ri = U_Rj  then  if |K_Ri| ≥ |K_Rj| then
                                   R = R - R_i; <<K_Ri and K_Rj are
                              else R = R - R_j;   keys of R_i and
      end;                                       R_j>>
   end;
end PART-2;
end algorithm;
```

(with key K_{R_i}) from the schema R, only if there exists a relation scheme R_j (with key K_{R_j}) in R such that:

(i) $R_i = R_j$,

or

(ii) $U_{R_i} = U_{R_j}$ and $|K_{R_i}| \geq |K_{R_j}|$.

The proof for the case of condition (i) is straightforward and needs no explanation. For the other case, suppose relation schemes R_i and R_j are constructed from dependencies F_i: $X \longrightarrow Y$ and f_j: $X' \longrightarrow Y'$ respectively. Since $X \cup Y = X' \cup Y'$ (by assumption), and if we assume $X' \subset X$, then f_i is embodied in $R_j(\underline{X}', Y')$ (which is the same as $R_i(\underline{X}', \alpha, Y)$), and thus its information will not be lost if we delete R_i from the schema. A similar discussion now can be given if we relax on the assumption $X' \subset X$, completing the proof. #

Although the new algorithm eliminates some of the problems discussed earlier, the nonredundancy and separation principles cannot be satisfied. The following examples clarify these concepts.

Example 3.3: In our University Housing data base, if each COMPLEX can specify a set of RENTs for its various apartments, and if a COMPLEX together with the APT-CAPACITY can determine the RENT of the apartment, then we may have the following dependencies:

g_1: COMPLEX,APT-CAPACITY $\longrightarrow$ RENT

m_1: COMPLEX $\longrightarrow\!\!\!\!\!\rightarrow$ RENT

and we can construct the following relation schemes:

R_1(COMPLEX, APT-CAPACITY, RENT)

R_2(COMPLEX, RENT)

Although relation scheme R_2 of the above schema is redundant,but it is not removed by the Algorithm 3-2, and thus violating nonredundancy principle. Indeed, the schema does not satisfy the separation principle either, since it is not even in 2NF (it is easy to verify that in relation scheme R_1, attribute RENT is partially dependent on the key).

Example 3.4: Again in our University Housing data base, if each TENANT lives in only one apartment (APT#) which has one TYPE and is in one COMPLEX, and if each COMPLEX has only one TYPE of apartments, then we may have the following functional dependencies:

f_1: TENANT $\longrightarrow$ APT#,COMPLEX,TYPE

f_2: COMPLEX $\longrightarrow$ TYPE

using the Algorithm 3-2, we can construct the following relation schemes:

R_1(TENANT, APT#, COMPLEX, TYPE)

R_2(COMPLEX, TYPE)

Notice that, in relation scheme R_1, attribute TYPE is transitively dependent on key TENANT, and thus violates the 3NF (although it is in 2NF).

The above examples indicate that the Algorithm 3-2 may produce a schema violating either nonredundancy principle or

separation principle (and in some cases both).

As a matter of fact, all of these problems arose mainly because we have completely ignored the composing rules for FDs and MVDs discussed in Chapter II. Thus, for further modifications of the algorithm these rules will be also considered.

3.5. The Dependency-to-Relation Method, 2nd Improvement

Considering the theoretical tools discussed in Chapter II, we can now further modify our algorithm. We will first modify the algorithm using the augmentation rule, and will prove that the schema constructed in this way is at least in second normal form (but not necessarily in 3NF or 4NF). Then we will have one more improvement to our method, this time considering the transitivity rule. We will see that these considerations eliminate many of the problems shown previously in our examples. Finally, we will present an algorithm considering all inference rules for both, FDs and MVDs. Then we will prove that the relational schema constructed by the algorithm satisfies all of the three design principles.

It is important to note, that in the algorithm which follows (Figure 3-3), only explicit dependencies (i.e., those FDs and MVDs given as part of the data base description) are considered, and therefore, not all of the problems may be eliminated by this algorithm. Thus, in Section 3.6 we will further modify the algorithm by considering not only explicit

FDs and MVDs, but also considering implicit dependencies (i.e., dependencies that are not explicitly presented as part of the data base description, but can be inferred from them by applying the inference rules) as well.

3.5.1. Proof of Representation

Theorem 3.4. Let R be a relational schema constructed from a set of dependencies G = F U M using the Algorithm 3-3, then R represents the same information as G.

Proof: The proof for the schema constructed by the first two parts of the algorithm follows from Theorem 3.3. Third part of the algorithm may remove either an extraneous attribute from a relation scheme of the schema, or an entire relation scheme of the schema. We follow the proofs for both cases.

(i). Removing extraneous attributes from a relation scheme: a set of attributes is removed from a relation scheme R_i only if it is a subset of non-key attributrs of another relation scheme R_j whose key is a subset of the key of relation R_i. Therefore, only those attributes are removed from a relation, that not only are represented in another relation, but their relationship/s are also represented in a stronger form than original relation.

(ii). Removing a relation scheme from the schema: Relation scheme R_i is removed from the schema R only if after removing its extraneous attributes it becomes the same as another scheme in the schema. Thus its elimination from the schema

Algorithm 3-3

An Algorithm to Construct a Schema from a Set of Dependencies; 2nd Improvement

INPUT: A set F of m FDs; and a set M of n MVDs.

OUTPUT: A set R of relation schemes (in 1NF or better).

PART-1: <<this part constructs one relation scheme for each dependency>>

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=i}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme;

end;

$$R'' = \bigcup_{i=m+1}^{m+n}$$

Figure 3-3.

Figure 3-3 (continued)

```
<<put all relation schemes together>>
   R = R' ∪ R";
end PART-1;
PART-2:  <<this part removes a relation R_i if it has the
           same set of attribures as R_j, and the relation-
           ships of its attributes are not as strong as
           R_j>>
     do for each R_i ε R  for 1 ≤ i ≤ r - 1; <<r is cardinality
        of R>>
          do for each R_j ε R  for i + 1 ≤ j ≤ r;
               if U_Ri = U_Rj then  if  |K_Ri| ≥ |K_Rj| then
                                        R = R - R_i; <<K_Ri and
                                            K_Rj are
                                    else  R = R - R_j; keys of
                                            R_i and R_j>>
          end;
       end;
   end PART-2;
PART-3:  <<this part eliminates explicit partial dependencies
           from relation schemes>>
    do for each R_i ε R';
       do for each R_j ε R' - R_i;
          if K_Rj ⊂ K_Ri then [
```

Figure 3-3 (continued)

$\Psi = (U_{R_i} - K_{R_i}) \cap (U_{R_j} - K_{R_j})$,

$U_{R_i} = U_{R_i} - \Psi$,

if $U_{R_i} = K_{R_i}$ then $R = (R - R_i) \cup R_i]$;

end;

end;

end PART-3;

end algorithm;

can be done without losing any information.

As the result, neither elimination of the extraneous attributes nor removal of the entire relation schemes (in PART-3), cause losing any information, completing the proof. #

Although the representation principle is not violated by the Algorithm 3-3 (Theorem 3.4), but the schema constructed in this way may not even be in 2NF, and thus violating separation principle. The following examples and discussions clarify this problem.

Example 3.5: Given the following dependencies:

f_1: A,B $\longrightarrow$ C,D

f_2: B $\longrightarrow$ D,E

the algorithm first constructs the following relation schemes:

R_1(<u>A</u>, <u>B</u>, C, D)

R_2(<u>B</u>, D, E)

and then removes the extraneous attribute D from R_1.

R_1(<u>A</u>, <u>B</u>, C)

R_2(<u>B</u>, D, E)

Although the schema of the above example is in 2NF (at least), but there are cases that the schema constructed by the Algorithm 3-3 is not necessarily in 2NF. This can simply be indicated by the following example.

Example 3.6: Given the following dependencies:

f_1: A,B $\longrightarrow$ C,D,E

f_2: B $\longrightarrow$ F
f_3: F $\longrightarrow$ D,G
f_4: A $\longrightarrow$ E

the algorithm first constructs one relation scheme for each dependency as follows:

R_1(A,B,C,D,E)
R_2(B,F)
R_3(F,D,G)
R_4(A,E)

and then removes the extraneous attribute E from R_1.

R_1(A,B,C,D)
R_2(B,F)
R_3(F,D,G)
R_4(A,E)

In the above schema, relation schemes R_2, R_3, and R_4 are in 2NF (at least). But how about R_1? R_1 is not in 2NF because, attribute D is partially dependent on key {A,B} (from f_2 and f_3, and by Axiom FD3 we get FD B $\longrightarrow$ D,G; and from FD B $\longrightarrow$ D,G and Axiom FD6 we get FD B $\longrightarrow$ D).

The above problems arose mainly because we have considered only those dependencies that are explicitly given, and not those implied dependencies that can be inferred from the original set by applying inference rules. This is considered in our next step of modifications.

3.6. The Dependency-to-Relation Method, 3rd Improvement

Recall the Closure II and Closure III of a set of data dependencies discussed in Chapter II. We need to consider them here for further improvements to our algorithm. In

addition, some subsets of these closures that are formally stated in the following definitions, are also of particular interest for further modifications.

Definition 3.1. Let G = F ∪ M be a set of mixed dependencies (FDs and MVDs), then FD-Closure II of G, denoted by FD-G^{II} is defined to be the set of all functional dependencies in G^{II}.

Definition 3.2. Let G F M be a set of mixed dependencies (FDs and MVDs), then FD-Closure III of G, denoted by FD-G^{III} is defined to be the set of all functional dependencies in G^{III}.

Considering the FD-Closure III of data dependencies, now we can present the Algorithm 3-4 (Figure 3-4) which as we will prove later produces schema at least in 2NF.

3.6.1. Proof of Representation

Theorem 3.5. Let R be a relational schema constructed from a set of dependencies G = F ∪ M using the Algorithm 3-4, then R represents the same information as G.

Proof: From Theorem 3.4, it follows that the schema constructed by the first three parts of the algorithm embodies F and M. PART-4 of the algorithm removes a set of attributes Ψ from a relation R_i only if (i) Ψ is in FD-G^{III}, which means that it is represented (implicitly) by some other relation schemes, and (ii) the relationship of Ψ with other attributes in those relations is stronger than the relationship in R_i. In other

Algorithm 3-4

An Algorithm to Construct a Schema From a Set of Dependencies; 3rd Improvement

INPUT: A set F of m FDs; a set M of n MVDs; and FD-$(F \cup M)^{III}$.

OUTPUT: A set R of relation schemes (in 2NF or better).

PART-1: <<this part constructs one relation scheme for each dependency>>

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=1}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme;

end;

$$R'' = \bigcup_{i=m+1}^{m+n} R_i;$$

Figure 3-4

Figure 3-4 (continued)
<<put all relation schemes together>>

```
   R = R' ∪ R";
end PART-1;
PART-2:  <<this part removes a relation R_i if it has the
           same set of attributes as R_j, and the relationships
           of its attributes are not as strong as R_j>>
   do for each R_i ε R  for 1 ≤ i ≤ r - 1; <<r is cardinality
            of R>>
      do for each R_j ε R  for i + 1 ≤ j ≤ r;
         if U_Ri = U_Rj  then if |K_Ri| ≥ |K_Rj|  then
                                   R = R - R_i; <<K_Ri and K_Rj are
                              else R = R - R_j;  keys of R_i and
      end;                                      R_j>>
   end;
end PART-2;
PART-3:  <<this part eliminates explicit partial dependencies
           from relation schemes>>
   do for each R_i ε R';
      do for each R_j ε R; - R_i;
         if K_Rj ⊂ K_Ri  then [
            Ψ = (U_Ri - K_Ri) ∩ (U_Rj - K_Rj),
            U_Ri = U_Ri - Ψ,
            if   U_Ri = K_Ri  then R = (R - R_i) ∪ R_i];
      end;
```

Figure 3-4 (continued)

```
   end;
end PART-3;
PART-4:  <<this part eliminates implicit partial dependencies
            from relation schemes>>
repeat:
   do for each g_j ε (FD-G^III - G);
      do for each R_i ε R';
         if LEFT_gj ⊂ K_R_i  then [
            Ψ = (U_R_i - K_R_i) ∩ RIGHT_g_j,
            U_R_i = U_R_i - Ψ,
            if  U_R_i = K_R_i  then R = (R - R_i) ∪ R_i,
            if Ψ = ∅  then  R' = R' ∪ R_h(ω), where ω = (LEFT_g_j
                      ∪ Ψ)
                                                  and LEFT g_j is
                                                  the key];
      end;
   end;
until no relation scheme is added to R';
end PART-4;
end algorithm;
```

words, only those attributes are removed from a relation scheme that not only are represented in other relation schemes of the schema, but their relationships are also represented (in a stronger form than in original relation) by them. Hence, in PART-4 of the algorithm no information would be lost, completing the proof. #

3.6.2. Proof of Normalization

The schema constructed by the Algorithm 3-4 is not in the desirable form (i.e., 4NF) yet, but it is in a better form than the schemas constructed previously. This claim is formalized by the following theorem.

Theorem 3.6. Let R be a relational schema constructed from a set of dependencies using the Algorithm 3-4, then R is in second normal form.

Proof: From Theorem 3.2, it follows that the schema constructed by the first part of the algorithm is in first normal form. Parts 3 and 4 of the algorithm eliminate all partial dependencies (both, explicit and implicit) from relation schemes, and thus leaving a 2NF schema. #

Algorithm 3-4 eliminates some of the problems discussed earlier, but not all of them. It produces a schema which is at least in 2NF, while the former algorithms could only guarantee the 1NF schemas. As it was stated in Chapter I, although a schema in 2NF has less undesirable properties than if it was in 1NF, but it is not yet completely free of all

undesirable properties, and thus, still violates the separation principle. The following examples clarify these concepts.

example 3.7: Given the following dependencies:

f_1: A,D $\longrightarrow$ B,C

f_2: A $\longrightarrow$ B,E

m_1: A $\longrightarrow\!\!\!\!\rightarrow$ D

the algorithm first constructs the following schemes:

R_1(A, D, B, C)

R_2(A, B, E)

R_3(A, D)

and then removes attribute B from R_1.

R_1(A, D, C)

R_2(A, B, E)

R_3(A, D)

and since FD A $\longrightarrow$ C holds in R_1 (see Figure 3-5), then attribute C is also partially dependent on key (in R_1), and thus is removed from R_1. Now, R_1(A, D) becomes the same as relation R_3 and hence it is removed. Finally, relation scheme R_4(A, C) is added to the schema by the algorithm.

R_2(A, B, E)

R_3(A, D)

R_4(A, C)

As the reader will notice, all partial dependencies are removed from the schemes, allowing the schema to be at

Derivation of $A \longrightarrow C$ from $A,D \longrightarrow B,C$ and $A \longrightarrow\!\!\!\!\rightarrow D$.

f_1:	$A,D \longrightarrow B,C$	assumption
f_1':	$A,D \longrightarrow C$	f_1 and axiom FD6
m_1':	$A,D \longrightarrow\!\!\!\!\rightarrow C$	f_1' and axiom FD-MVD1
m_1:	$A \longrightarrow\!\!\!\!\rightarrow D$	assumption
m_1'':	$A \longrightarrow\!\!\!\!\rightarrow C$	m_1, m_1', and axiom MVD4
f_1'':	$A \longrightarrow C$	m_1'', f_1', and axiom FD-MVD2

Figure 3-5.

least in 2NF (in fact, the above schema is in 4NF).

Example 3.8: Given the following dependencies:

f_1: A $\longrightarrow$ B,C,D

f_2: B $\longrightarrow$ C

the algorithm produces the following two schemes:

R_1(A, B, C, D)

R_2(B, C)

As we can see, the schema is in 2NF but not in 3NF. This is becuase, in relation scheme R_1 attribute C is transitivly dependent on key A (it is dependent on attribute B). This problem can be overcome by decomposing this type of relation schemes into their projections, using the transitive dependencies. This issue will be considered in the following section.

3.7. The Dependency-to-Relation Method, 4th Improvement

In this section we will further modify our algorithm to construct a schema at least in 3NF. To do so, we need to detect those dependencies that exist between two sets of non-prime attributes (or between a set of non-prime attributes and a proper subset of attributes making the key) in a relation scheme. This enables us to decompose the relation scheme into two of its projections, and thus eliminating many undesirable properties (if not all of them) from the schema. This is performed in PART-5 of the Algorithm 3-5 (Figure 3-6).

Algorithm 3-5

An Algorithm to Construct a Schema From a Set of Dependencies; 4th Improvement

INPUT: A set F of m FDs; a set M of n MVDs; and FD-$(F \cup M)^{III}$.

OUTPUT: A set R of relation schemes (in 3NF or better).

PART-1: <<this part constructs one relation scheme for each dependency>>

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=1}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme:

end;

$$R'' = \bigcup_{i=m+1}^{m+n} R_i;$$

Figure 3-6.

Figure 3-6 (continued)

```
<<put all relation schemes together>>
   R = R' ∪ R";
end PART-1;
PART-2:  <<this part removes a relation R_i if it has the
           same set of attributes as R_j, and the relation-
           ships of its attributes are not as strong as R_j>>
   do for each R_i ε R  for 1 ≤ i ≤ r - 1; <<r is cardinality
             of R>>
      do for each R_j ε R  for i + 1 ≤ j ≤ r;
         if U_Ri = U_Rj  then  if |K_Ri| ≥ |K_Rj|  then
                                  R = R - R_i; <<K_Ri and K_Rj
                                                 are
                               else R = R = R_j;   keys of R_i and
                                                   R_j>>
      end;
   end;
end PART-2;
PART-3:  <<this part eliminates explicit partial dependencies
           from relation schemes>>
   do for each R_i ε R';
      do for each R_j ε R' - R_i;
         if K_Rj ⊂ K_Ri  then [
            Ψ = (U_Ri - K_Ri) ∩ (U_Rj - K_Rj),
            U_Ri = U_Ri - Ψ,
            if   U_Ri = K_Ri then R = (R - R_i) ∪ R_i];
```

Figure 3-6 (continued)

```
      end;
   end;
end PART-3;
PART-4:  <<this part eliminates implicit partial dependencies
           from relation schemes>>
repeat:
   do for each g_j ε (FD-G^III - G);
      do for each R_i ε R';
         if LEFT_g_j ⊆ K_R_i  then [
            Ψ = (U_R_i - K_R_i) ∩ RIGHT_g_j,
            U_R_i = U_R_i - Ψ,
            if  U_R_i = K_R_i  then R = (R - R_i) ∪ R_i,
            if Ψ = ∅  then  R' = R' ∪ R_h(ω), where ω = LEFT_g_j
                       ∪ Ψ)
                                                  and LEFT_g_j is
                                                  the key];
      end;
   end;
until no relation scheme is added to R';
end PART-4;
PART-5:  <<this part eliminates transitive dependencies
           from relation schemes>>
repeat:
   do for each R_i ε R';
```

Figure 3-6 (continued)

```
      Ω = U_Ri - K_Ri  <<Ω is the set of non-key attributes>>
      do for each g_j ε FD-G^III;
         if U_gj ⊆ Ω then [decompose R_i into
         R_i1 (U_gj)  with LEFT_gj as the key,
   and   R_i2 (K_Ri U (Ω-RIGHT_gj))  with K_Ri as the key,
         R' = (R - R_i) U R_i1 U R_i2 ];
      end;
   end;
until no relation scheme is added to R':
end PART-5;
end algorithm;
```

3.7.1. Proof of Representation

Existence of an MVD in a relation is a necessary and sufficient condition for that relation to be decomposable into two of its projections without loss of information. That is, the original relation can be reconstructed by joining (natural join) those two projections. This is proved by Fagin in [18]. He has presented the following theorem.

Theorem 3.7. MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ holds for relation $R(X,Y,Z)$ if and only if R is the join of its projections $R_1(X,Y)$ and $R_2(X,Z)$.

Proof: It is simple to verify that $R(X,Y,Z)$ is the join of its projections $R_1(X,Y)$ and $R_2(X,Z)$ if and only if the following condition holds: Whenever (x,y,z) and (x,y',z') are tuples of R, then so are (x,y',z) and (x,y,z'). This latter condition holds iff $Y_{xz} = Y_{xz'}$. The proof now follows from the definition of MVDs. #

From Fagin's theorem we can conclude the following corollary.

Corollary 3.1. The existence of a data dependency, FD or MVD, in a relation is a necessary and sufficient condition for that relation to be decomposable to its projections without loss of information.

Proof: The proof follows from Theorem 3.7 and the axiom FD-MVD1 which is: if an FD $X \longrightarrow Y$ holds for R, then so does MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$. #

Considering Theorem 3.7 and Corollary 3.1, now we can

have the following theorem.

Theorem 3.8. Let R be a relational schema constructed from a set of dependencies G = F U M using the Algorithm 3-5, then R represents the same information as G.

Proof: From Theorem 3.5 it follows that the relational schema constructed by the first four parts of the algorithm represents F and M. Fifth part of the algorithm decomposes a relation R_i into two of its projections R_{i_1} and R_{i_2} based on an FD (or an MVD) holding in it. Therefore, from Corollary 3.1 it follows that the original relation R_i can be reproduced from R_{i_1} and R_{i_2} using the natural join. Thus, R_{i_1} and R_{i_2} represent the same information represented by R_i completing the proof. #

3.7.2. Proof of Normalization

Proposition 3.1. If a relational schema is in nth Normal Form (nNF) for $2 \leq n \leq 4$, then the projection operation on relations never can convert the schema into a worse Normal Form, that is mNF for $m < n$.

Proof: The proof follows from the definition of Normal Forms (separation principle), and from the definition of projection operation. #

Theorem 3.9. Let R be a relational schema constructed from a set of dependenceis using the Algorithm 3-5, then R is in Third normal form.

Proof: From Theorem 3.6 it follows that the schema

constructed by the first four parts of the algorithm is in 2NF. And from Proposition 3.0 it follows that the fifth part of the algorithm cannot worsen the normalized form of the schema, thus leaving it to be still at least in 2NF. Now, because the schema is in 2NF, and since no transitive dependency exists in any relation scheme (those dependencies are all removed by PART-5), thus the schema is in 3NF, completing the proof. #

While Algorithm 3-5 eliminate all of the problems. The following examples clarify these concepts.

Example 3.9: Given the following FDs:

f_1: A $\longrightarrow\!\!\!\!\rightarrow$ B,D,E,F
f_2: B $\longrightarrow$ C
f_3: C $\longrightarrow$ D
f_4: E $\longrightarrow$ F

the algorithm first constructs the following relation schemes:

R_1(<u>A</u>, B, D, E, F)
R_2(<u>B</u>, C)
R_3(<u>C</u>, D)
R_4(<u>E</u>, F)

then it decomposes R_1 into R_{11}(<u>E</u>, F) and R_{12}(<u>A</u>, B, D, E), and removes R_{11} since it is the same as R_4.

R_{12}(<u>A</u>,B,D,E)
R_2 (<u>B</u>,C)
R_3 (<u>C</u>,D)
R_4 (<u>E</u>,F)

and finally, since FD B $\longrightarrow$ D is in FD-G^{III} (it can be derived from f_2 and f_3 by applying axiom FD3), the algorithm decomposes R_{12} into $R_{121}(\underline{B},D)$ and $R_{122}(\underline{A},B,E)$:

$R_{121}(\underline{B},D)$

$R_{122}(\underline{A},B,E)$

$R_2(\underline{B},C)$

$R_3(\underline{C},D)$

$R_4(\underline{E},F)$

As we can see the schema is in 3NF (it is in fact, in 4NF).

Example 3.10: Given the following dependencies:

f_1: A $\longrightarrow$ B,C,D

m_1: D $\longrightarrow\!\!\!\!\rightarrow$ B

the algorithm first constructs the following relations:

$R_1(\underline{A},B,C,D)$

$R_2(\underline{D},\underline{B})$

then, it decomposes R_1 into $R_{11}(\underline{D},B)$ and $R_{12}(\underline{A},C,D)$ because FD D $\longrightarrow$ B holds in R_1 (from FD A $\longrightarrow$ B,C,D and Axiom FD6 we can get FD A $\longrightarrow$ B; from MVD D $\longrightarrow\!\!\!\!\rightarrow$ B and FD A $\longrightarrow$ B and Axiom FD-MVD2 we can get FD D $\longrightarrow$ B).

$R_{11}(\underline{D},B)$

$R_{12}(\underline{A},C,D)$

$R_2(\underline{D},\underline{B})$

The only problem with this schema is, that the relation $R_2(\underline{D},\underline{B})$ is now redundant and must be removed. In fact, we have already removed this kind of redundant relation schemes

in PART-2 of the algorithm. Thus, to eliminate the above problem we can apply PART-2 of the algorithm for any new relation scheme constructed by the other parts of the algorithm. This consideration is shown in our Final algorithm.

Example 3.11: Given the following dependencies:

f_1: A $\longrightarrow$ B,C,D,E

f_2: A,B $\longrightarrow$ C

f_3: D $\longrightarrow$ E

m_1: F $\longrightarrow\!\!\!\!\rightarrow$ G,E

the algorithm first constructs the following relations:

R_1(<u>A</u>,B,C,D,E)

R_2(<u>A</u>,<u>B</u>,C)

R_3(<u>D</u>,E)

R_4(<u>F</u>,<u>G</u>,<u>E</u>)

then it removes attribute C from R_2

R_1(<u>A</u>,B,C,D,E)

R_2(<u>A</u>,<u>B</u>)

R_3(<u>D</u>,E)

R_4(<u>F</u>,<u>G</u>,<u>E</u>)

now R_1 is decomposed into R_{11}(<u>D</u>,E) and R_{12}(<u>A</u>,B,C,D) of which R_{11} is deleted, because it is the same as R_3.

R_{12}(<u>A</u>,B,C,D)

R_2(<u>A</u>,<u>B</u>)

R_3(<u>D</u>,E)

R_4(<u>F</u>,<u>G</u>,<u>E</u>)

Note, the problem with this schema is not only that $R_2(\underline{A},\underline{B})$ is redundant, but also $R_4(\underline{F},\underline{G},E)$ is not in 4NF (because, from FD f_3 and MVD m_1 and axiom FD-MVD2 it follows that $F \longrightarrow E$).

To overcome the above problems we need to further modify our algorithm. This, which will be in fact, the final modification regarding the construction of a "desirable" schema is stated in the following section.

3.8. The Dependency-to-Relation Method, Final Improvement

Now, we are in the final state of the iterative refinement of the algorithm. The algorithm presented in this section (Figure 3-7), constructs a schema which will be proved to be in 4NF (hereafter, we will call this algorithm the FNF Algorithm). All proofs concerning the FNF Algorithm are given in Chapter V. In that chapter, we will formally examine the properties of the schema produced by the FNF Algorithm, and will propose directions for constructing optimal schemas.

The proofs of representation and normalization (and also nonredundancy) for the FNF Algorithm will be given in Chapter V. Here, we only give an example to show that the problems discussed earlier are eliminated by the final algorithm. The data base of the following example is taken from [18].

Example 3.12: Suppose we have a university data base with

Algorithm 3-6

An Algorithm to Construct a Fourth Normal Form Schema from a Set of Dependencies: (FNF Algorithm)

INPUT: A set F of m FDs; a set M of n MVDs; the set $(F \cup M)^{II}$; the set $(F \cup M)^{III}$; and the set $FD-(F \cup M)^{III}$.

OUTPUT: A set R of relation schemes in 4NF.

PART-1: <<this part constructs one relation scheme for each dependency>>

<<construct relation schemes from FDs>>

do for each $f_i \in F$;

construct a relation scheme R_i consisting of all attributes appearing in f_i; The set of attributes that appears on the left side of f_i is the key of the relation scheme;

end;

$$R' = \bigcup_{i=1}^{m} R_i;$$

<<construct relation schemes from MVDs>>

do for each $m_i \in M$;

construct a relation scheme R_{i+m} consisting of all attributes appearing in m_i; The set of all attributes that appears in m_i is the key of the relation scheme;

end;

Figure 3-7

Figure 3-7 (continued)

$$R'' = \bigcup_{i=m+1}^{m+n} R_i$$

<<put all relation schemes together>>

R = R' U R";

end PART-1;

PART-2: <<this part removes a relation R_i if it has the same set of attributes as R_j, and the relationships of its attributes are not as strong as R_j>>

do for each $R_i \in R$ for $1 \leq i \leq r - 1$; <<is cardinality of R>>

do for each $R_j \in R$ for $i + 1 \leq j \leq r$;

if $U_{R_i} = U_{R_j}$ then if $|K_{R_i}| \geq |K_{R_j}|$ then

$R = R - R_i$; <<K_{R_i} and K_{R_j} are keys of R_i and R_j>>

else $R = R - R_j$;

end;

end;

end PART-2;

PART-3: <<this part eliminates explicit partial dependencies from relation schemes>>

do for each $R_i \in R'$;

do for each $R_j \in R' - R_i$;

if $K_{R_j} \subset K_{R_i}$ then [

Figure 3-7 (continued)

$\Psi = (U_{R_i} - K_{R_i}) \cap (U_{R_j} - K_{R_j})$,
$U_{R_i} = U_{R_i} - \Psi$
if $U_{R_i} = K_{R_i}$ then $R = (R - R_i) \cup R_i$] ;

end;

end;

end PART-3;

PART-4: <<this part eliminates implicit partial dependencies from relation schemes>>

repeat:

do for each $g_j \in (FD\text{-}G^{III} - G)$;

do for each $R_i \in R'$;

if $LEFT_{g_j} \subset K_{R_i}$ then [
$\Psi = (U_{R_i} - K_{R_i}) \cap RIGHT_{g_j}$,
$U_{R_i} = U_{R_i} - \Psi$,
if $U_{R_i} = K_{R_i}$ then $R = (R - R_i) \cup R_i$,
if $\Psi = \emptyset$ then $R' - R' \cup R_h(\omega)$, where $\omega = (LEFT_{g_j} \cup \Psi)$ and $LEFT_{g_j}$ is the key];

end;

end;

until no relation scheme is added to R';

end PART-4;

Figure 3-7 (continued)

PART-5: <<this part eliminates transitive dependencies from relation schemes>>

repeat:

do for each $R_i \in R'$;

$\Omega = U_{R_i} - K_{R_i}$ << Ω is the set of non-key attributes>>

do for each $g_j \in FD\text{-}G^{III}$;

if $U_{g_j} \subseteq \Omega$ then [decompose R_i into

$R_{i_1}(U_{g_j})$ with $LEFT_{g_j}$ as the key,

and $R_{i_2}(K_{R_i} \cup (\Omega - RIGHT_{g_j}))$ with K_{R_i} as the key,

$R' = (R' - R_i) \cup R_{i_1} \cup R_{i_2}$];

end;

end;

until no relation scheme is added to R':

end PART-5;

PART-6: <<this part decomposes decomposable schemes constructed from FDs>>

repeat:

do for each $R_i \in R'$;

$\Omega = U_{R_i} - K_{R_i}$, <<$\Omega$ is the set of non-key attributes>>

do for each $g_j \in G^{III}$;

if $LEFT_{g_j} \subseteq \Omega$ and $RIGHT_{g_j} \subset K_{R_i}$ then [decompose R_i into $R_{i_1}(RIGHT_{g_j} \cup LEFT_{g_j})$ with $LEFT_{g_j}$ as the key if:

Figure 3-7 (continued)

$g_j \in FD\text{-}G^{III}$, and

($RIGHT_{g_j} \cup LEFT_{g_j}$) otherwise;

and R_{i_2} ($LEFT_{g_j} \cup (U_{R_i} - RIGHT_{g_j})$) with all attributes appearing in R_{i_2} as the key;

$R' = (R' - R_i) \cup R_{i_1} \cup R_{i_2}$]

end;

end;

until no relation scheme is added to R';

end PART-6;

PART-7: <<this part decomposes decomposable schemes constructed from MVDs>>

repeat:

do for each $g_j \in G^{II}$;

do for each $R_i \in R''$;

if ($LEFT_{g_j} \cup RIGHT_{g_j}$) $\subset R_i$ then [decompose R_i into

R_{i_1} ($LEFT_{g_j} \cup RIGHT_{g_j}$) with $LEFT_{g_j}$ as the key if:

$g_j \in FD\text{-}G^{II}$, and

($RIGHT_{g_j} \cup LEFT_{g_j}$) otherwise;

and R_{i_2} ($U_{R_i} - RIGHT_{g_j}$) with all attributes appearing in R_{i_2} as the key;

$R'' = (R'' - R_i) \cup R_{i_1} \cup R_{i_2}$];

end;

end;

until no relation scheme is added to R'';

repeat PART-2 for the new set $R = R' \cup R''$;

Figure 3-7 (continued)

```
end PART-7;
PART-8: <<this part removes those redundant schemes
         that are represented (implicitly) by other
         schemes of the schema>>
   do for each R_i ε R;
      do for each g_j ε (G^II - G);
         if (LEFT_g_j U RIGHT_g_j )=U_R_i and LEFT_g_j = K_R_i then
               R = R - R_i;
      end;
   end;
end PART-8;
end algorithm;
```

attributes CLASS, SECTION, STUDENT, MAJOR, EXAM, YEAR, INSTRUCTOR, RANK, SALARY, TEXT, DAY, and ROOM. If a given CLASS consists of several SECTIONS, each of which has one INSTRUCTOR and various STUDENTs. And if each CLASS has a set of TEXTs, which are used by all SECTIONs of the CLASS. Also assume, that each SECTION of a CLASS meets on various DAYs, and on a given DAY, it has one meeting ROOM. We also know that, each STUDENT has only one MAJOR, and one YEAR. A STUDENT in a given CLASS and SECTION has several EXAM scores. And if each INSTRUCTOR has one RANK and one SALARY, then we can have the following FDs and MVDs:

f_1: CLASS,SECTION $\longrightarrow$ INSTRUCTOR

f_2: CLASS,SECTION,DAY $\longrightarrow$ ROOM

f_3: STUDENT $\longrightarrow$ MAJOR,YEAR

f_4: INSTRUCTOR $\longrightarrow$ RANK,SALARY

m_1: CLASS,SECTION $\longrightarrow\!\!\!\!>$ STUDENT,MAJOR,EXAM,YEAR

m_2: CLASS,SECTION $\longrightarrow\!\!\!\!>$ INSTRUCTOR,RANK,SALARY

m_3: CLASS,SECTION $\longrightarrow\!\!\!\!>$ DAY,ROOM

m_4: CLASS $\longrightarrow\!\!\!\!>$ TEXT

m_5: CLASS,SECTION,STUDENT $\longrightarrow\!\!\!\!>$ EXAM

the FNF Algorithm first produces the following relation schemes (one scheme for each dependency):

R_1(<u>CLASS</u>, <u>SECTION</u>, INSTRUCTOR)

R_2(<u>CLASS</u>, <u>SECTION</u>, <u>DAY</u>, ROOM)

R_3(<u>STUDENT</u>, MAJOR, YEAR)

R_4(INSTRUCTOR, RANK, SALARY)

R_5(CLASS, SECTION, STUDENT, MAJOR, EXAM, YEAR)

R_6(CLASS, SECTION, INSTRUCTOR, RANK, SALARY)

R_7(CLASS, SECTION, DAY, ROOM)

R_8(CLASS, TEXT)

R_9(CLASS, SECTION, STUDENT, EXAM)

then it decomposes R_5 into

R_{51}(STUDENT, MAJOR, YEAR) and

R_{52}(CLASS, SECTION, STUDENT, EXAM)

and it decomposes R_6 into

R_{61}(CLASS, SECTION, INSTRUCTOR) and

R_{62}(CLASS, SECTION, RANK, SALARY)

and since R_{51} is the same as R_3, R_{52} is the same as R_9, and R_{61} is the same as R_1, they are all removed from the schema. Relation scheme R_{62} is also removed from the schema because it is represented by R_1 and R_2 (in an stronger form). For the same reason, R_7 which is represented by R_2 in a stronger form, is deleted. Therefore, the final schema constructed by the algorithm is:

R_1(CLASS, SECTION, INSTRUCTOR)

R_2(CLASS, SECTION, DAY, ROOM)

R_3(STUDENT, MAJOR, YEAR)

R_4(INSTRUCTOR, RANK, SALARY)

R_8(CLASS, TEXT)

R_9(CLASS, SECTION, STUDENT, EXAM)

As reader will notice the constructed schema is in fourth normal form (the proof and related formal discussions are given in Chapter V).

3.9. Chapter Summary and Remarks

A new design approach for constructing relational schemas from dependencies (FDs and MVDs) designated by the data base administrator, has been proposed and investigated. After an iterative refinement of the approach, a final algorithm (FNF) which produces 4NF relation schemes has been presented. It has been proved that the schema constructed by this method satisfies the three design principles (the proofs concerning the FNF Algorithm will be given in Chapter V), and thus it is free from undesirable properties discussed in Chapter I.

CHAPTER IV

CONSTRUCTING CLOSURE II AND CLOSURE III

4.1 Introduction

In this chapter two algorithms are presented for constructing Closure II and Closure III of a given set of data dependencies. Also, detailed analyses of these algorithms are given, and investigations are made to indicate that the worst case (i.e., the case in which all data dependencies designated by the data base administrator are completely chained together) of each algorithm is unlikely to happen for real world applications. Anyhow, to maintain the consistency of the design approach, this special case must also be considered. An algorithm which detects this case and warns the data base administrator of the "bad" situation is given at the end of this chapter.

4.2 Description of the Closure II Algorithm

An FD f: X $\longrightarrow$ Y is in canonical form if and only if |Y| = 1, that is, the right side of the function consists of one attribute [8]. A set of FDs is in canonical form if and only if all of its member FDs are in canonical form.

Proposition 4.1. For any set of FDs, there exists an equivalent set (i.e., a set representing the same information)

of FDs in canonical form.

Proof: Any FD f: $X \longrightarrow Y$ ($Y = \{Y_1, Y_2, \ldots, Y_n\}$, where $n > 1$) can easily be converted to a set of FDs $F = \{f_1, f_2, \ldots, f_n\}$ in canonical form using Axiom FD6.

f_1: $X \rightarrow Y_1$,

f_2: $X \rightarrow Y_2$,

.
.
.

f_n: $X \rightarrow Y_n$;

completing the proof. #

Considering proposition 4.1, we will assume without loss of generality that the set of FDs used as input to our Closure II Algorithm (Figure 4-1) is in canonical form. But, a similar assumption cannot be made for the set of MVDs, simply because having an MVD m:$X \twoheadrightarrow \{Y_1, Y_2, \ldots, Y_n\}$, does not necessarily result to

m_1: $X \twoheadrightarrow Y_1$,

m_2: $X \twoheadrightarrow Y_2$,

.
.
.

m_n: $X \twoheadrightarrow Y_n$.

We will also assume that for every FD $X \rightarrow Y$ (or MVD $X \twoheadrightarrow Y$) used as input, $Y \not\subseteq X$. This is a reasonable assumption for real world applications because, FD $X \rightarrow X$ (or MVD $X \twoheadrightarrow X$) means, having a set of attributes X, we can determine (or multidetermine) a set of attributes X, and clearly, having a set X at our disposal we do not

Algorithm 4-1

An Algorithm to construct the Closure II of a set of data dependencies

INPUT: A set F of FDs in canonical form; a set M of MVDs; and $G = F \bigcup M$.

OUTPUT: Closure II of G, G^{II}; FD-G^{II}; and MV-G^{II}.

```
D = J' = F;
V = J" = M;
  repeat
    H' = ∅;
    H" = ∅;
      do for each f_i ε F+M;
         do for each f_j ε J'+J";
            if RIGHT_f_i ⊆ LEFT_f_j then
               [if f_j ε J'   [construct an FD f_k with
                               LEFT_f_k = LEFT_f_i ⋃ (LEFT_f_j - RIGHT_f_i) and
                               RIGHT_f_k = RIGHT_f_j;
                  if RIGHT_f_j ⊄ LEFT_f_k  then
                     H' = H ⋃ f_k;
                  else;];
                else [construct an MVD f_k with
                      LEFT_f_k = LEFT_f_i ⋃ (LEFT_f_j = RIGHT_f_i) and
                      RIGHT_f_k = RIGHT_f_j - LEFT_f_j;
                  if RIGHT_f_k ⊆ LEFT_f_k  then
                     H" = H" ⋃ f_k]];
```

Figure 4-1

Figure 4-1 continued

```
            end;
        end;
        D = D⋃H';
        V = V⋃H";
        J' = H';
        J" = H";
    until H' + H" = ∅;
    FD-G^II = D;
    MV-G^II = V;
    G^II = D + V;
end algorithm;
```

have to determine (or multidetermine) it. Anyhow, those kind of data dependencies must be detected (if any) to avoid inconsistency of the approach. A detection algorithm is presented at the end of this chapter.

4.2.1 Proof of termination

Theorem 4.1. Algorithm 4-1 halts.

Proof: Both, inner loop and innermost loop of the algorithm are finite loops because F + M and J' + j" are finite sets. At each iteration of outer loop, the innermost loop generates a number of new data dependencies. Since the cardinality of the set generated at pass i (for $2 \leq i \leq n$) is less than the cardinality of the set generated at pass i - 1 (because, for the worst case in which all dependencies are chained together, if p dependencies are generated at pass i - 1, the number of dependencies generated at pass i is p - 1), then the algorithm ultimately reaches to the point that H' + H" = $\emptyset$, and hence it halts. #

4.2.2 Proof of correctness

Theorem 4.2. Upon termination of the Algorithm 4-1, G^{II} is the Closure II of G, FD-G^{II} contains all FDs in G^{II}, and MV-G^{II} contains all MVDs in G^{II}.

Proof: As we can observe, G^{II} is calculated by the algorithm as $G^{II} = G + \bigcup_{i=1}^{n} (H' + H")_i$, where $(H' + H")_i$ is the set of dependencies generated at pass i. Having G at the

first pass, the set $(H' + H")_1$ generated, is in fact G^2, and similarly set $(H' + H")_{n-1}$ generated at pass n - 1 is G^n†. Therefore, we have $G^{II} = G \cup G^2 \cup G^3 \cup \ldots \cup G^n = \bigcup_{i=1}^{n} G^i$.

Now, the proof of the correctness for this formula will be similar to the case that G^{II} is the transitive closure of G [28]. The proof is in two parts.

1) $\bigcup_{i=1}^{n} G^i \subset G^{II}$, where n is the smallest integer satisfying $G^n \cup G^{n+1} = G^n$. We prove this part by induction.

(basis)

From Definition 2.1 (given in Chapter II), it follows that $G \subset G^{II}$.

(induction step)

Suppose $G^i \subset G^{II}$, $i \geq 1$, and let $\langle A,B \rangle \in G^{i+1}$, where $\langle A,B \rangle$ means $A \rightarrow B$ (or $A \rightarrow\!\!\!\!\rightarrow B$). Since $G^{i+1} = G^i G$, there exists some $C \in U$ such that $\langle (A - C),(C - A) \rangle \in G^i$ and $\langle C,B \rangle \in G$. By the induction hypothesis and the basis step, $\langle (A - C), (C - A) \rangle \in G^{II}$ and $C,B \in G^{II}$. Since G^{II} is closed under axioms FD4 and MVD4, it follows that $\langle A,B \rangle \in G^{II}$, thus completing the induction.

2) $G^{II} \subset \bigcup_{i=1}^{n} G^i$. Let $\langle A,B \rangle$ and $\langle B,C \rangle$ be arbitrary elements of $\bigcup_{i=1}^{n} G^i$. Then for some positive integers p and q, $\langle A,B \rangle \in G^p$ and $\langle B\alpha,C \rangle \in G^q$. Then $\langle A\alpha ,C \rangle \in G^p G^q$ (or $\langle A\alpha,C \rangle \in G^{p+q}$).

---

† The concept is borrowed from the following definition:
Def. Let R be a binary relation on a set A and let $n \in N$. Then, R^n is defined as follows:

1. R^0 is the relation of equality on the set A.
2. $R^{n+1} = R^n R$

Therefore, $<A\alpha,C> \in \bigcup_{i=1}^{n} G^i$ and hence $\bigcup_{i=1}^{n} G^i$ contains all dependencies derivable from G using Axioms FD4 and MVD4, completing the proof of the first part of the theorem.

We claim also that the type of data dependency (FD or MVD) is considered by the algorithm appropriately. In fact, for each pair of data dependencies f_i and f_j to be checked, four possibilities should be considered:

1) Both f_i and f_j belong to the family of FDs. For this case the generated dependency would be of type FD.

2) f_i belongs to FDs family, and f_j belongs to MVDs family. Since f_i belongs to FDs family, then it also belongs to MVDs family (Axiom FD-MVD1). Thus, both f_i and f_j are MVDs, and the generated dependency is also an MVD.

3) Both f_i and f_j belong to MVDs family. As for the $case_2$ the generated data dependency would be of type MVD.

4) f_i belongs to MVDs family, and f_j belongs to FDs family. Again, since f_j is an FD, it is also an MVD, and thus we can generate a data dependency f_m of type MVD whose right side is the same with the right side of f_j. Now, by applying FD-MVD2 to the pair f_m and f_j we can generate an FD f_k, because (i) $LEFT_{f_j} \subseteq LEFT_{f_m}$ (in fact, we showed that $LEFT_{f_j} = LEFT_{f_m}$), and (ii) $RIGHT_{f_m} \cap LEFT_{f_j} = \emptyset$ (by assumption). Therefore, the

resulting dependency for this case is of type FD.

From the above discussion we conclude that the generated data dependency is of type FD if and only if f_j belongs to the set of FDs, and it is an MVD otherwise. This is considered in innermost loop of the algorithm (the statement, if $f_j \in J'$ ...), thus completing the proof. #

4.2.3 Output analysis

At each iteration of the outer loop, a new data dependency (either FD or MVD) is generated if and only if there is a connection (in the sense of Axioms FD4 and MVD4) between a pair of dependencies. More connections between pairs of dependencies exists, more new data dependencies are generated. Thus, the best case is in fact when there is no connection between any pair, and therefore, output is only as large as input. Suppose there are m FDs and n MVDs as the input to the algorithm, then $|G^{II}| = |F| + |M| = m+n$. On the other hand, the worst case happens when all given data dependencies are completely chained together. For this case, the number of data dependencies generated by the algorithm is proportional to $(m+n)^2$, which is proved by the following theorem.

Theorem 4.3. Let G^{II} be the Closure II of $G = (F \bigcup M)$ constructed by the Algorithm 4-1, then for the worst case (all dependencies chained together), $|G^{II}| = (m+n)(m+n+1)/2$, where $m = |F|$ and $n = |M|$.

Proof: By induction:

(m + n = 1)

If only one data dependency is involved, no new data dependency is generated by the algorithm (because the process requires at least a pair of dependencies), and thus the output consists of only 1 data dependency, that is, the original one. Using the above formula for m + n = 1, we also get $|G^{II}| = 1$.

(induction step)

If for i data dependencies $|G_i^{II}| = i \cdot (i+1)/2$, and if we add a new data dependency to our input set, and since by our assumption of the worst case, all i data dependencies have connections with the (i + 1)th data dependency, thus i more dependencies are generated. This has to be added to 1 (for the added dependency itself) to get the total number of data dependencies, $|G_{i+1}^{II}| = i \cdot (i + 1)/2 + (i + 1) = (i + 1) \cdot ((i + 1) + 1)/2$, thus completing the induction. #

As we saw, the formula of Theorem 4.3, is for a situation in which all data dependencies designated by data base administrator are chained together. As an example, consider the following case where G is

f_1: $A_1 \rightarrow A_2$,

f_2: $A_2 \rightarrow A_3$,

f_3: $A_3 \rightarrow A_4$,

⋮

f_i: $A_i \rightarrow A_{i+1}$,

$\vdots$

f_m: $A_m \rightarrow A_{m+1}$,

m_1: $A_{m+1} \rightarrow\!\!\!\!\rightarrow A_{m+2}$,

m_2: $A_{m+2} \rightarrow\!\!\!\!\rightarrow A_{m+3}$,

$\vdots$

m_j: $A_{m+j} \rightarrow\!\!\!\!\rightarrow A_{m+j+1}$,

$\vdots$

m_n: $A_{m+n} \rightarrow\!\!\!\!\rightarrow A_{m+n+1}$.

Now, suppose there is a "gap" (to cut the chain) somewhere in the chain, say, close to the middle of the set, then we have two subsets each with the approximate length of (m+n)/2 and both in the worst case (all elements of each subset are fully chained together). Then to find the number of elements in G^{II}, we can use the formula of Theorem 4.3 for each subset and then multiply the result by 2. Similarly, for 2 gaps in the set, we get 3 subsets, and we need to compute the number of data dependencies generated in one of them and multiply it by 3 (assuming that all subsets are of the same length). By the same token, if there are j gaps in the set, we get

$$|G^{II}| = \frac{(m+n)\ (m+n+j+1)}{2(j+1)} \qquad (4.2.2.1)$$

To have a better understanding of the formula (4.2.2.1) for the real world applications, we should notice that, if (m+n) is not very large (say < 100),

then even if J is small, $|G^{II}|$ won't be too large. And for the large value of (m+n), J will be comparatively large too (from the fact that not most of the dependencies in a large data base are fully chained), and again $|G^{II}|$ won't be too large.

4.2.4 Speed analysis

At the first pass through the outer loop, and for each iteration of the inner loop, the innermost loop executes (m+n) times (m=|F|, and n = |M|), and since inner loop itself is bounded by m + n iterations, then the entire loop executes $(m+n)^2$ times. If no new dependency is generated by this pass (i.e., the best case -- no connections between any pair of dependencies), then algorithm terminates in time $0(p^2)$, where p = m+n. Considering the worst case, the outer loop iterates p times (at each iteration the number of dependencies generated is one dependency less than the previous iteration), and hence, the overall time for the algorithm is below $0(p^3)$.

(best case) $T_b = p^2$

(worst case) $T_w = p.(p)+p.(p-1)+p.(p-2)+\ldots+1 = p^2.(p+1)/2$

4.3 Description of the Closure III Algorithm

All assumptions for this algorithm are the same as what we had for the Algorithm 4-1. The basic objective of the algorithm is to construct G^{III} -- the

Closure III of a given set of data dependencies G, using Axioms FD4, MVD4, and FD-MVD2. Since G^{II}, which is the result of applying Axioms FD4 and MVD4 to the set G, can be constructed by the Algorithm 4-1, then the algorithm 4-2 gets as input G^{II} of G, and applies Axiom FD-MVD2 to G^{II} to get G^{III} (See Figure 4.2).

Example 4.1: This carefully selected example indicates the significance of the outer loop of the algorithm. The proofs of correctness and termination are given later.

Suppose the following set of data dependencies are given as the input to Algorithm 4-2:

m_1: $A \rightarrow\!\!\!\!\rightarrow \{B,C,D\}$,

m_2: $E \rightarrow\!\!\!\!\rightarrow \{B,F\}$,

m_3: $G,H \rightarrow\!\!\!\!\rightarrow \{B,I\}$,

f_1: $F,C,E \rightarrow \{B\}$.

then, at the first pass the algorithm generates the FD

f_{11}: $G,H \rightarrow \{B\}$

and then using FD f_{11}, another FD is generated

f_{12}: $E \rightarrow \{B\}$

and finally, using FD f_{12}, the algorithm generates

f_{13}: $A \rightarrow \{B\}$.

4.3.1. Proof of correctness

Theorem 4.4. Upon termination of the Algorithm 4-2, all FDs derivable from the set G^{II} by applying FD-MVD2 are in FD-G^{III}, and thus G^{III} is the Closure III of G.

Algorithm 4-2

An Algorithm to construct the Closure III of a set of data dependencies

INPUT : MV-G^{II} and FD-G^{II} of a set of data dependencies G;

OUTPUT: G^{III} and FD-G^{III} of G;

$V = MV\text{-}G^{II}$;

$D = FD\text{-}G^{II}$;

$FD\text{-}G^{III} = FD\text{-}G^{II}$;

repeat

$H = V' = \emptyset$;

do for each $f_i \in V$;

do for each $f_j \in D$;

if $RIGHT_{f_j} \subseteq RIGHT_{f_i}$ then

[if $RIGHT_{f_i} \cap LEFT_{f_j} = \emptyset$ then [construct an FD f_k with

$LEFT_{f_k} = LEFT_{f_i}$ and

$RIGHT_{F_K} = RIGHT_{f_j}$;

if $RIGHT_{f_k} \not\subseteq LEFT_{f_k}$ then

$H = H \cup f_k$;

else;]

else $V' = V' \cup f_i$;]

end;

end;

Figure 4-2

Figure 4-2 continued

```
        V = V';
        D = H;
        FD-G^III = FD-G^III ⋃ H;
    until H = ∅  or V' = ∅;
    G^III = FD-G^III + MV-G^II;
end algorithm;
```

Proof: At the first pass of the algorithm, all MVDs in the set are checked with all FDs (pair wise) and the new FDs are generated. For the following passes, not all of the MVDs of the original set are needed to be checked with new FDs, and therefore, algorithm considers only those MVDs that have a chance of generating new FDs. This can be formalized as follows:

Suppose the input to the algorithm is

f_1: $X_1 \rightarrow Y_1$

f_2: $X_2 \rightarrow Y_2$

$\vdots$

f_j: $X_j \rightarrow Y_j$

$\vdots$

f_m: $X_m \rightarrow Y_m$

m_1: $W_1 \twoheadrightarrow Z_1$

m_2: $W_2 \twoheadrightarrow Z_2$

$\vdots$

m_j: $W_j \twoheadrightarrow Z_j$

$\vdots$

m_n: $W_n \twoheadrightarrow Z_n$

where sets X's, Y's, W's and Z's are not necessarily disjoint. At pass i, an MVD $W_j \twoheadrightarrow Z_j$ is removed and will not be considered for pass i+1, if either

1) $Y_i \not\subseteq Z_j$ for $1 \leq i \leq m$,

or

2) for all $y_i \subseteq Z_j$ X_i and Z_j are disjoint -- $X_i \cap Z_j = \emptyset$

Part (1): If the set H of new FDs generated by pass i is

f_{i1}: $W'_1 \rightarrow Y'_1$

f_{i2}: $W'_2 \rightarrow Y'_2$

$\vdots$

f_{ip}: $W'_p \rightarrow Y'_p$,

then the existence of MVD $W_j \twoheadrightarrow Z_j$ in the set of MVDs in pass i+1 is not significant, and it does not generate any newer Functional Dependency. In fact, if any FD is generated from MVD $W_j \twoheadrightarrow Z_j$ and the set H, then it must be an FD $W_j \rightarrow Y'_k$ (for some value of k in $1 \leq k \leq p$). This can happen only <u>if</u> $Y'_k \subseteq Z_j$ (Axiom FD-MVD2), and it is a contradiction to our assumption because we have

$$Y'_k \in \{Y'_1, Y'_2, \ldots, Y'_p\} \qquad \text{and}$$

$$\{Y'_1, Y'_2, \ldots, Y'_p\} \subseteq \{Y_1, Y_2, \ldots, Y_m\}$$

and thus

$$Y'_k \in \{Y_1, Y_2, \ldots, Y_m\}$$

and by our assumption, no member of $\{Y_1, Y_2, \ldots, Y_m\}$ can be a subset of Z_j.

Part (2): Again, to prove the redundancy of the MVD $W_j \twoheadrightarrow Z_j$ for pass i+1, we need to prove that its existence does not cause generating any newer FD. In fact, if MVD $W_j \twoheadrightarrow Z_j$ and the new set H of FDs, can generate a newer functional dependency, then it must be FD $W_j \rightarrow Y'_k$ (for some value of k in $1 \leq k \leq p$). This is because there must be an FD $W'_k \rightarrow Y'_k$ such that $Y'_k \subseteq Z_j$ and $W'_k \cap Z_j = \emptyset$. If $Y'_k \subseteq Z_j$, then since we have

$Y'_k \in \{Y'_1, Y'_2, \ldots, Y'_p\}$ and

$\{Y'_1, Y'_2, \ldots, Y'_p\} \subseteq \{Y_1, Y_2, \ldots, Y_m\}$

then we get

$Y'_k = Y_h$ for some value of h in $1 \leq h \leq m$.

And by our assumption, for all $Y_i \subseteq Z_j$ (including Y_h), we have $X_i \cap Z_j = \emptyset$. Thus MVD $W_j \twoheadrightarrow Y'_k$ could be generated in pass i, and there is no need to carry MVD $W_j \twoheadrightarrow Z_j$ to pass i+1, completing the proof. #

4.3.2 Proof of termination

Theorem 4.5 Algorithm 4-2 halts.

Proof: Both inner loop and innermost loop of the algorithm are finite loops because V and D are finite sets for all iterations of the outer loop. Since at each iteration of the outer loop either set H or set V (sometimes both) becomes smaller, then the algorithm always halts. #

4.3.3 Output Analysis

Proposition 4.2 Let F be a set of FDs and M be a set of MVDs, then the maximum possible number of FDs that can be generated from F and M using Axion FD-MVD2 is $|F| . |M|$.

Proof: Consider the following data dependencies:

f_1: $X_1 \rightarrow Y_1$

f_2: $X_2 \rightarrow Y_2$

⋮

f_m: $X_m \rightarrow Y_m$

m_1: $W_1 \twoheadrightarrow Z_1$

m_2: $W_2 \twoheadrightarrow Z_2$

$\vdots$

m_n: $W_n \twoheadrightarrow Z_n$

Any FD $W' \rightarrow Y'$, generated from the above set by applying Axiom FD-MVD2 must have the following properties:

$W' \in \{W_1, W_2, \ldots, W_n\}$ and

$Y' \in \{Y_1, Y_2, \ldots, Y_m\}$.

Now, since there are n possibilities for W' and m possibilities for Y', then there are m.n (i.e., $|F|.|M|$) possibilities for the FD $W' \rightarrow Y'$. #

4.3.4 Speed analysis

At each iteration of the inner loop, innermost loop executes (at most) m times ($m = |FD\text{-}G^{II}|$). At each iteration of the outer loop, inner loop iterates n times ($n = |MV\text{-}G^{II}|$). Therefore, at each iteration of the outer loop, the innermost loop executes m.n times. Considering proposition 4.2, and assuming that each single new functional dependency is generated at one iteration of the outer loop (the worst case), then the outer loop iterates m.n times and thus the overall time for the algorithm is $O(m.n)^2$. For the best case which most likely happens for real world applications, all expected FDs are generated at the first pass of the algorithm, and thus the algorithm

halts in time O(m.n).

4.4 Description of the Detection Algorithms

As we mentioned earlier, there are some special cases that will never happen for the real world applications, when the design issue of the relational data base is of interest. Although we believe in this fact, we still care about the consistency of the approach, and thus those special cases will be detected (if they happen). Two algorithms that detect the two most unusual cases (in the sense of data dependencies designated by the data base administrator), and warn the data base administrator of the "bad" situations are presented in this section. Algorithm 4-3 (Figure 4-3) discovers the "fully chained data dependencies" situation, and Algorithm 4-4 (Figure 4-4) detects the "reflexive" data dependencies.

4.5 Chapter Summary and Remarks

Two algorithms for computing Closure II and Closure III of a given set of data dependencies have been presented. It has been shown that the worst case of these algorithms is unlikely to happen for real world applications. In spite of this fact, to maintain the consistency of the method, two algorithms have been presented to detect these worst cases (if they happen) and warn the data base administrator of the unexpected situation.

Algorithm 4-3

An Algorithm to detect the fully chained data dependencies situation

INPUT : A set F of m FDs; and a set M of n MVDs;

OUTPUT: "YES", if all data dependencies in ($F \bigcup M$) are chained together, and "NO" otherwise;

STEP 1: (Initialization) m = k = i = 1; count = 0;

STEP 2: (Get the right side of dependency) right=$RIGHT_{gi}$; j = 1;

STEP 3: (Check if all dependencies are chained) if count = n - 1 then print "YES", STOP;

STEP 4: (Check if this dependency has been tested) if i = j or MARK(i) = 1 then go to STEP 7;

STEP 5: (Check the left side of dependency) if right $\not\subseteq$ $LEFT_{gj}$ or MARK(j) = 1 then go to STEP 7;
PUSH [right,j,i] into STACK; MARK(i) = 1;
count = count + 1; i = j; go to STEP 2;

STEP 6: (Pop from stack). if STACK empty go to STEP 8;
POP STACK in [right,j,i]; MARK(i)=0; count = count - 1;

STEP 7: (Check if all dependencies have been tested for this right side) if j < n then j = j + 1, go to STEP 3; else go to STEP 6;

STEP 8: (Termination) if m < n then m = m + 1, i = m, go to STEP 2; else print "NO", STOP;

Figure 4-3

Algorithm 4-4

An Algorithm to detect the reflexive data dependencies

INPUT : A set F of m FDs; and a set M of n MVDs;

OUTPUT: A set $X \subseteq (F \bigcup M)$ of reflexive data dependencies;

```
X = ∅;
do for each f_i ε F + M;
   if RIGHT_fi ⊆ LEFT_fi then
      X = X ⋃ f_i;
end;
end algorithm;
```

Figure 4-4

CHAPTER V

ANALYSIS OF RELATIONS BY DEPENDENCY-TO-RELATION

5.1 Introduction

In Chapter III an algorithm (FNF Algorithm) was presented for constructing a relational schema from a set of functional dependencies and multivalued dependencies. In this chapter properties of the relation schemes constructed by the FNF Algorithm are discussed (section 5.2), and it is shown that although the schema constructed is desirable (i.e., contains no undesirable properties), it is not necessarily "optimal". Heuristics for choosing "reasonable" decompositions which lead to an "optimal" schema are suggested, and an algorithm which decomposes a set of decomposable schemes into an optimal form is given. Finally, two previous approaches are examined in sections 5.3 and 5.4, and they are compared with the new approach.

5.2 Properties of Relations constructed by FNF Algorithm

Recall the three design principles (Representation, Separation, and Nonredundancy) given in Chapter II. It is important to characterize the properties of the relational schema constructed by the FNF Algorithm to see if all three design principles are satisfied, and thus, to determine whether the schema is an "ideal" schema or not.

5.2.1 Representation Principle

The schema constructed by our FNF Algorithm, clearly, must not violate the representation principle. It must represent all information of interest (i.e., the same information as input). Different researchers have defined the concept of "the same information" in different ways. Four different definitions are discussed and compared in [6]. In this section, we first describe the most common and reasonable definition, and we will show that the schema constructed by the FNF Algorithm satisfies the representation principle (assuming this definition). Then representation is defined in another way, a definition which our schema of FNF Algorithm does not always satisfy. These alternatives are studied, and suggestions are given to the possible resolution of the paradox.

Definition 5.1. Let R and S be two relational schemas, then S represents the same information as R if they have the same attributes and databases of S contain the same data as the databases of R.

The above definition which relys on the corollary 3.1 given in Chapter III, can be clarified by the following examples.

Example 5.1: Suppose an schema R consists of a relation scheme TACT(TENANT,ATP#,COMPLEX,TYPE), and in addition to FD TENANT $\longrightarrow$ APT#,COMPLEX,TYPE,the FD COMPLEX $\longrightarrow$ TYPE holds in TACT, and suppose an instance of relation TACT is as in Figure 5-1a. Now, if the schema S consists of two relation

schemes CT(COMPLEX,TYPE) and TAC(TENANT,APT#,COMPLEX) which are indeed, the projections of R on COMPLEX and TYPE; and on TENANT, APT#, and COMPLEX, then an instance of S is as in Figure 5-1b. As the reader will notice, the schema S bears the same information that the schema R does, and thus by definition 5.1 schemas R and S are equivalent. This is in fact because the projection operation has been performed based on a data dependency, that is, the FD COMPLEX $\longrightarrow$ TYPE (corollary 3.1). The following example indicates a situation in which the projections of a relation carry more information (in fact, nonsense information) than the original relation does.

Example 5.2: Consider the schema R of Example 5.1 and its snap-shot given in Figure 5-1a. Now, if a schema S' consists of the projections of R on TYPE and APT#; and on TYPE, TENANT, and COMPLEX, then an instance of S' is as in Figure 5-2. As we see in Figure 5-2 each TENANT is associated with not only his APT but also with some other APTs of some other TENANTs. That is, the schema S' does not represent the same information as schema R, and thus assuming Definition 5.1 for representation, they are not equivalent.

Considering Definition 5.1, the FNF Algorithm always constructs a schema which satisfies the representation principle. This claim is formalized in the following theorem.

Theorem 5.1. Let R be a schema constructed from a set of data dependencies G, using the FNF Algorithm, then R represents G.

An example of a relation and its projections

a) An instance of the schema R.

TACT(TENANT,	APT#,	COMPLEX,	TYPE)
Jack	17D	Nieman	Fur.
Mary	212H	Parkview	Unfur.
Phil	305F	Kraettli	Fur.
Mark	302A	Kraettli	Fur.
Mike	208C	Kraettli	Fur.
John	126H	Const.	Unfur.
Tom	113F	Const.	Unfur.

b) An instance of the schema S (projections of R).

CT(COMPLEX,	TYPE)
Neiman	Fur.
Parkview	Unfur.
Kraettli	Fur.
Const.	Unfur.

TAC(TENANT,	APT#,	COMPLEX)
Jack	17D	Nieman
Mary	212H	Parkview
Phil	305F	Kraettli
Mark	302A	Kraettli
Mike	208C	Kraettli
John	126H	Const.
Tom	113F	Const.

Figure 5-1.

Proof: PART-1 of the algorithm constructs one relation scheme for each FD and one relation scheme for each MVD. Therefore, the schema constructed in PART-1 represents G. In PART-2, only those schemes that represent the same information as some other schemes of the schema are deleted, and thus the schema of PART-2 represents the schema of PART-1 which in turn represents G. PART-3 and PART-4 of the algorithm remove only extraneous attributes (i.e., attributes represented elsewhere in the schema, in stronger form) from relation schemes; and thus no information is lost in these two PARTs, and the schema constructed here (at the end of PART-4) represents the schema of PART-2. The other three parts of the algorithm, that is, PART-5, PART-6, and PART-7 decompose all of the decomposable schemes of the schema. All decomposition steps in these PARTs are based on the data dependencies holding in relation schemes, and thus considering Corollary 3.1, and assuming Definition 5.1, the decomposed schema (i.e., the final schema of the algorithm) represents the schema of PART-4 which in turn represents G, completing the proof. #

Definition 5.2. Let R and S be two relational schemas, then S represents the same information as R if it contains the same attributes and the same data dependencies as R.

Considering Definition 5.2, the FNF Algorithm does not always construct an schema satisfying the representation principle. In fact, the schema constructed by the first four PARTs of the algorithm satisfies the representation principle

An instance of the schema S'.
(different projections of R of Figure 5-1a).

AT(APT#,	TYPE)	TTC(TENANT,	TYPE,	COMPLEX)
17D	Fur.	Jack	Fur.	Nieman
212H	Unfur.	Mary	Unfur.	Parkview
305F	Fur.	Phil	Fur.	Kraettli
302A	Fur.	Mark	Fur.	Kraettli
208C	Fur.	Mike	Fur.	Kraettli
126H	Unfur.	John	Unfur.	Const.
113F	Unfur.	Tom	Unfur.	Const.

Figure 5-2

(Theorem 3.5), but PART-5, PART-6, and PART-7 may decompose relation schemes into subrelations violating some of the data dependencies. The following example clarifies this concept.

Example 5.3: Given the following data dependencies G,

f_1: $A,B \longrightarrow C,D,E$

m_1: $C \longrightarrow\!\!\!\!\rightarrow B$

the algorithm first constructs the following schemes,

$$S_1 \begin{cases} R_1(\underline{A,B},C,D,E) \\ R_2(\underline{C,B}) \end{cases}$$

then it decomposes R1 into $R_{11}(\underline{C,B})$ and $R_{12}(\underline{A,C,D,E})$, and removes R11 because it is the same as R_2.

$$S_2 \begin{cases} R_{12}(\underline{A,C,D,E}) \\ R_2(\underline{C,B}) \end{cases}$$

Nctice that the schema S_1 represents the same information as G (assuming either concepts -- Def. 5.1, or Def. 5.2), and the schema S_2 represents S_1 only if Definition 5.1 is considered. In other words, the databases of S_2 contain the same data as the databases of S_1 (satisfying Def. 5.1), but the FD $A,B \longrightarrow C,D,E$ is no longer represented by the schema S_2 (violating Def. 5.2).

One may suggest that instead of converting the schema S_1 into S_2, we could simply remove the relation scheme R2 from S_1, and thus, the schema S_1 consisting of R_1 satisfies both definitions of representation. The problem is, that as we discussed in Chapter II, the representation principle is not

the only principle that has to be satisfied by a "good" schema. The above suggestion, in fact, ignores the separation principle, and thus allows the schema to have undesirable properties.

One possible solution to the problem is to associate the dependency which is lost by a decomposition process to corresponding subrelations, by means of storing it in a specified area. This area then can be looked over when those subrelations need to be updated. Notice that the proposed solution can no longer be expedient if the above situation (i.e., losing a data dependency in a decomposition process) happens frequently in the schema design.

5.2.2 Separation Principle

Another goal in designing a "good" schema is to construct a schema in which the basic "units of information" are represented separately from each other. This is in fact, the intuitive motivation behind the normalization process [13]. In other words, the intention is to remove the undesirable properties (i.e., those properties that cause update anomalies discussed in Chapter I) from the schema by reducing the information to its more basic units. In this section, we will show that the schema constructed by the FNF Algorithm, is free from undesirable properties, and indeed it is in 4NF. This claim is formally illustrated by the following propositions and theorem.

Proposition 5.1. Considering a 2NF relation scheme R(P,N) where P is the primary key of R, and N is a non-empty set, then the MVD m: $X \twoheadrightarrow Y$ (or FD f: $X \rightarrow Y$) does not hold in R if both X and Y are subsets of P.

Proof: (By contradiction)

Since X and $Y \subset P$, then relation R can be represented as R(X,Y,A,N), that is, we have

f_1: $X,Y,A \rightarrow N$

from f_1 and Axiom FD-MVD1 we get

m_1: $X,Y,A \twoheadrightarrow N$

and if m: $X \twoheadrightarrow X$ holds in R, then from m and m_1 and Axiom MV04 we get

m_3: $X,A \twoheadrightarrow N$

from m_3 and f_1 and Axiom FD-MVD2 we have

f_2: $X,A \rightarrow N$

which means N is partially dependent on key. This partial dependency is a contrary to the fact that the relation R is in 2NF.

Proposition 5.2. Given a 3NF relation scheme R(P,N) where p is the primary key of the relation R, and N is a non-empty set, then the MVD m: $X \twoheadrightarrow Y$ (or FD f: $X \rightarrow Y$) does not hold in R if both X and Y are subsets of N.

Proof: (By contradiction)

By our assumption relation R can be indicated as R(P,X,Y,B) from which we can conclude

f_1: $P \rightarrow \{X,Y,B\}$

from f and Axiom FD6 we get

f_2: $P \longrightarrow Y$

from f and m (assuming it does hold in R), and Axiom FD-MVD2 we get

f_3: $X \longrightarrow Y$

which means attribute Y is transitively dependent on key, and thus violating 3NF schema characteristics, and contradiction to our assumption. #

Proposition 5.3. If relation scheme R(P,N) where P is the primary key of R and N is a non-empty set, is in 2NF, and if $X \subset P$ and $Y \subset N$, then MVD m: $X \longrightarrow\!\!> Y$) (or FD f_i $X \longrightarrow$) does not hold in R. Proof: (By contradiction)

If P = {X,A} and N = {Y,B} then we have R(X,A, Y,B) and we get

f_1: $X,A \longrightarrow \{Y,B\}$

from f_1 and Axiom FD6 we get

f_2: $X,A \longrightarrow Y$

from m (assuming it does hold in R) and f_2, and Axiom FD-MVD2 we get

f_3: $X \longrightarrow Y$

which means a non-prime attribute is partially dependent on key, and thus violating the fact that R is in 2NF. #

Theorem 5.2. Let R be a relational schema constructed using the FNF Algorithm, then R is in 4NF.

Proof: From Theorem 3.9 it follows that the schema constructed by the first five PARTs of the algorithm is at least in 3NF. PART-6 and PART-7 of the algorithm decompose any relation

scheme in which a nontrivial multivalued dependency such as $X \longrightarrow\!\!\!\!\!\rightarrow Y$ holds, but not the functional dependency $X \longrightarrow C$ for every attribute C of that relation. This process which guarantees the schema to be in 4NF can be formalized as follows:

First, consider the set of relation schemes R' (that is, those relation schemes in which not all of the attributes make the key), and suppose $R_i(\underline{P},N) \in R'$ where P is a set of attributes making the key of R_i. Now, assuming the MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ holds for R_i, the following possibilities should be considered:

(1) X and Y $\in$ P

(2) X and Y $\in$ N

(3) X $\in$ P and Y $\in$ N

(4) X $\in$ N and Y $\in$ P

Possibilities (1), (2), and (3) are not acceptable according to propositions (5.1), (5.2), and (5.3) respectively. For the case (4), relation R_i can be indicated as $R_i(\underline{Y},\underline{A},X,B)$. Since MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ is an element of the set G^{III} and $X \subseteq \Delta$ (i.e., X is a non-prime attribute), and $Y \subseteq K_{R_i}$ (i.e., a subset of key), then PART-6 of the algorithm decomposes R_i into R_{i_1} and R_{i_2} based on MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$. Now, MVD $X \longrightarrow\!\!\!\!\!\rightarrow Y$ holds in R_{i_1}, and it is no longer a non-trivial MVD (because $X \cup Y = U_{Ri1}$).

By the above discussion we conclude that the set of relation schemes R' constructed by the first five PARTs of the algorithm is either already in 4NF, or it is converted

to 4NF schema by PART-6.

Second, any relation scheme $R_j \in R''$ (i.e., those relation schemes constructed merely from MVDs, and thus in each of them the entire set of attributes forms the key) is decomposed into its projections by PART-7 of the algorithm if a nontrivial MVD holds in it. Therefore, remaining relation schemes of R" are in 4NF, and thus the relational schema R which is $R' \cup R''$ is in 4NF. #

5.2.3. Minimal Redundancy Principle

By the minimal redundancy principle (sometimes called nonredundancy principle) it is meant that although the constructed schema must represent all of the information of interest, it should not contain any redundant information (we will discuss the concept of redundancy later in this section).

As for the representation principle, different researchers have defined minimal redundancy in different ways. Here in this section, we will present and discuss our own definition of minimality which embodies a quite different view from those given by others. As a matter of fact, in other definitions, it is the relation scheme upon which attention is focused, to check whether it is required for the schema, or it is redundant. On the other hand, in our definition of minimality, we place stress on attributes rather than schemes. In fact, our intention is to avoid the repetition

of attributes in the schema as much as possible, and we call the schema with this characteristic optimal. The interesting point is, that the optimal schema can be achieved by our approach. We will return to this point after the following formal discussions of minimality. We begin with a standard definition given in literature, and then present our own definition of minimality.

Definition 5.3 [6] Let R be a schema, then R is minimal if it does not contain any relation scheme R_i whose data dependencies are represented by the other schemas in R.

Definition 5.4 Let R be a schema, then R is minimal if for any other schema such as S which represents the same information as R, we have

$$\Sigma|R_i| \quad \leq \quad \Sigma|S_i|$$

for all $R_i \in R$ for all $S_i \in S$

Example 5.4: Given the following schema S:

$S\{R(\underline{A},\underline{B},\underline{C},\underline{D},\underline{E},\underline{F},\underline{G})\}$

and suppose two nontrivial MVDs m_1: $E \rightarrow\!\!\!\!\rightarrow G$ and m_2: $C,D \rightarrow\!\!\!\!\rightarrow F$ hold in R, then R can be decomposed into either $R_1(\underline{E},\underline{G})$ and $R_2(\underline{A},\underline{B},\underline{C},\underline{D},\underline{E},\underline{F})$, or $R_1'(\underline{C},\underline{D},\underline{F})$ and $R_2'(\underline{A},\underline{B},\underline{C},\underline{D},\underline{E},\underline{G})$ depending on the dependency chosen (m_1 or m_2) for decomposition. Therefore, the final schema will be either S' or S" (below).

$$\begin{cases} R_1(\underline{E},\underline{G}) \\ R_2(\underline{A},\underline{B},\underline{C},\underline{D},\underline{E},\underline{F}) \end{cases}$$

$$\begin{cases} R_1'(\underline{C},\underline{D},\underline{F}) \\ R_2'(\underline{A},\underline{B},\underline{C},\underline{D},\underline{E},\underline{G}) \end{cases}$$

Notice that while both S' and S" represent the same information

as S, but only S' is the minimal schema. In fact, two attributes C and D are repeated in schema S", while in schema S' only one attribute E is repeated twice, and thus we have

$$\Sigma|S_i'| \quad \text{for all } S_i' \in S' \qquad = |R_1| + |R_2| = 6 + 2 = 8$$

$$\Sigma|S_i''| \quad \text{for all } S_i'' \in S'' \qquad |R_1'| \quad |R_2'| = 6 + 3 = 9$$

In the above example, $\Sigma|S_i'|$ is less than $\Sigma|S_i''|$, because we decomposed S using MVD m_1 to get the schema S', and using MVD m_2 to get the schema S"; and this happened because $|LEFT_{m_1}| < |LEFT_{m_2}|$. This concept is formalized in the following proposition.

Proposition 5.4. Let R be the only decomposable relation scheme (i.e., some nontrivial data dependencies hold in it) in a schema S, and suppose there are alternatives for decomposing R (i.e., there are more than one nontrivial dependency holding in R), then decomposition leads to an optimal schema if the data dependency with the smallest number of attributes on its left side is chosen for decomposition.

Proof: Suppose one of the dependencies holding in $R(\Delta)$ is m: $X_j \twoheadrightarrow Y_j$, and suppose we decide to decompose R based on m, then we have

$$S' \quad \begin{cases} R_1(X_j, Y_j) \\ R_2(X_j, \Gamma) \text{ where } = \Gamma - \Delta X_j - Y_j \end{cases}$$

and thus we get

$$\Sigma|S_i'| \text{ for all } S_i' \in S' = |R_1| + |R_2| = |X_j| + |Y_j| + |X_j| + |\Gamma|$$

$$\Sigma|S_i'| \text{ for all } S_i' \in S' = |X_j| + |Y_j| + |X_j| + |\Delta| - |X_j| - |Y_j|$$

$$\Sigma|S_i'| \text{ for all } S_i' \in S' = |X_j| + |\Delta| \qquad (5.2.3.1)$$

and since $|\Delta|$ is constant, then the value of $\Sigma|S_i'|$ depends only on $|X_j|$, and thus it is minimum if $|X_j|$ is minimum, completing the proof. #

Notice that Proposition 5.4 assumes a schema with only one decomposable scheme. Indeed, if there are more than one decomposable schemes in a schema, then choosing a data dependency with the smallest left side does not necessarily lead to an optimal schema. The following example clarifies this concept.

Example 5.5: Given the following data dependencies:

m_1: A,B,D,E,F $\longrightarrow\!\!\!\!\!\rightarrow$ C,G

m_2: D,E,F $\longrightarrow\!\!\!\!\!\rightarrow$ I,J

m_3: D,E $\longrightarrow\!\!\!\!\!\rightarrow$ F

m_4: J $\longrightarrow\!\!\!\!\!\rightarrow$ E

we can first construct the following schema:

R_1 (A,B,D,E,F,C,G)

R_2 (D,E,F,I,J)

$R_3(\underline{D},\underline{E},\underline{F})$

$R_4(\underline{J},\underline{E})$

Here, relation schemes R_1 nad R_2 are decomposable; No alternatives exist for R_1, and it can be decomposed only based on dependency m_3. Thus, the projections are $R_{11}(\underline{D},\underline{E},\underline{F})$ and $R_{12}(A,B,D,E,C,G)$, and since R_{11} is the same as R_3 it is remoced, and we have

$R_{12}(\underline{A},\underline{B},\underline{D},\underline{E},\underline{C},\underline{G})$

$R_2(\underline{D},\underline{E},\underline{F},\underline{I},\underline{J})$

$R_3(\underline{D},\underline{E},\underline{F})$

$R_4(\underline{J},\underline{E})$

For R_2, there are two alternatives, it can be decomposed either based on dependency m_3 or dependency m_4. If we decompose R_2 based on dependency m_4 (with only one attribute on the left) into $R_{21}(\underline{J},\underline{E})$ and $R_{22}(\underline{D},\underline{F},\underline{I},\underline{J})$, and remove R_{21} which is the same as R_4, then we get

$$S' \quad \begin{cases} R_{12}(\underline{A},\underline{B},\underline{D},\underline{E},\underline{C},\underline{G}) \\ R_{22}(\underline{D},\underline{F},\underline{I},\underline{J}) \\ R_3(\underline{D},\underline{E},\underline{F}) \\ R_4(\underline{J},\underline{E}) \end{cases}$$

But, if we decompose R_2 based on dependency m_3 (with two attributes on the left) into $R_{21}'(\underline{D},\underline{E},\underline{F})$ and $R_{22}'(\underline{D},\underline{E},\underline{I},\underline{J})$, and remove R_{21}' which is the same as R_3, then we get

$R_{12}(\underline{A},\underline{B},\underline{D},\underline{E},\underline{C},\underline{G})$

$R_{22}'(\underline{D},\underline{E},\underline{I},\underline{J})$

$R_3(\underline{D},\underline{E},\underline{F})$

$R_4(\underline{J},\underline{E})$

and now, by decomposing R_{22}' (no alternative) into R_{22}' $1(\underline{J},\underline{E})$ and R_{22}' $2(\underline{D},\underline{I},\underline{J})$, and then removing $R_{22}'1$ which is the same as R_4 we get

$$S'' \quad \begin{cases} R_{12}(\underline{A},\underline{B},\underline{D},\underline{E},\underline{C},\underline{G}) \\ R_{22}'2(\underline{D},\underline{I},\underline{J}) \\ R_3(\underline{D},\underline{E},\underline{F}) \\ R_4(\underline{J},\underline{E}) \end{cases}$$

Notice that, although we didn't choose the dependency with the smallest left side to get the schema S", but S" is optimal. This is because

$$\Sigma|S_i''| = 14 \quad \text{for all } S_i'' \in S''$$

$$\Sigma|S_i'| = 15 \quad \text{for all } S_i' \in S'$$

and thus

$$\Sigma|S_i''| < \Sigma|S_i'| \quad \text{for all } S_i'' \in S \quad \text{for all } S_i' \in S'$$

This happened because, in decomposition process of S', we used two different data dependencies to decompose R_1 and R_2, while both relation schemes could be decomposed based on

the same data dependency (MVD m_2), and thus generating the same schemes that could be removed from the schema (in fact, it is what we did for constructing S"). This claim can be formally illustrated as follows:

Theorem 5.3. Let S be an schema containing some decomposable schemes, then decomposition leads to an optimal schema if:

(1) each decomposable scheme that has no dependency in common with other schemes, is decomposed based on the dependency with the smallest left side, and

(2) all decomposable schemes that have some dependencies in common, are decomposed based on the common dependency with the smallest left side.

Proof: The proof for the part(1) of the theorem directly follows from Proposition 5.4, and part(2) can be proved as follows:

Suppose S is a schema, and suppose $R \subseteq S$ with $|R| = p$ is a subschema consisting merely of the schemes that have some data dependencies holding in them in common, then using one of the common data dependencies, say, m: $X_j \twoheadrightarrow Y_j$, we can decompose them as follows:

$$
R \begin{cases} R_1(\Delta_1) & \xrightarrow{X_j \longrightarrow\!\!\!\!\!\rightarrow Y_j} \begin{cases} R_{11}(X_j, Y_j) \\ R_{12}(X_j, Y_1) \end{cases} \text{ where } \Gamma 1 = \Delta_1 - (X_j + Y_j) \\ R_2(\Delta_2) & \xrightarrow{X_j \longrightarrow\!\!\!\!\!\rightarrow Y_j} \begin{cases} R_{21}(X_j, Y_j) \\ R_{22}(X_j, {}_2) \end{cases} \text{ where } \Gamma 2 = \Delta_2 - (X_j + Y_j) \\ \vdots & \\ R_p(\Delta_p) & \xrightarrow{X_j \longrightarrow\!\!\!\!\!\rightarrow Y_j} \begin{cases} R_{p1}(X_j, Y_j) \\ R_{p2}(X_j, \Gamma_p) \end{cases} \text{ where } \Gamma_p = \Delta_p - (X_j + Y_j) \end{cases}
$$

Now, since all relation schemes R_{i1} (for $1 \leq i \leq p$) are the same, we remove all but R_{11} from the schema, and therefore we have

$$
R' \begin{cases} R_{11}(X_j, Y_j) \\ R_{12}(X_j, \Gamma_1) \\ R_{22}(X_j, \Gamma_2) \\ R_{32}(X_j, \Gamma_3) \\ \vdots \\ Rp_2(X_j, \Gamma_p) \end{cases}
$$

and hence, we get

$$
\begin{aligned}
\Sigma|R'_i| \quad \text{for all } R'_i \in R' \quad &= |X_j| + |Y_j| \\
&+ |X_j| + |\Delta_1| - |X_j| - |Y_j| \\
&+ |X_j| + |\Delta_2| - |X_j| - |Y_j| \\
&+ |X_j| + |\Delta_3| - |X_j| - |Y_j| \\
&\vdots \\
&+ |X_j| + |\Delta_p| - |X_j| - |Y_j|
\end{aligned}
$$

$$
\begin{aligned}
\Sigma|R'_i| \quad \text{for all } R'_i \in R' \quad &= (|\Delta_1 + \Delta_2 + \Delta_3 + \ldots + \Delta_p)| + \\
&\quad |X_j| - (p-1)\cdot|Y_j| \\
&= \sum_{m=1}^{p} |\Delta_n| + |X_j| - (p-1)\cdot|Y_j|
\end{aligned}
$$

(5.2.3.2)

On the other hand, if we decompose each relation scheme in R, using any arbitrary dependency not in common with other schemes, then we get

$$
R' \quad \begin{cases}
R_1(\Delta_1) \xrightarrow{A_1 \twoheadrightarrow B_1} \begin{matrix} R_{11}(A_1, B_1) \\ R_{12}(A_1, \Gamma'_1) \end{matrix} & \text{where } \Gamma'_1 = \Delta_1 - (A_1 + B_1) \\
R_2(\Delta_2) \xrightarrow{A_2 \twoheadrightarrow B_2} \begin{matrix} R_{21}(A_2, B_2) \\ R_{22}(A_2, \Gamma'_2) \end{matrix} & \text{where } \Gamma'_2 = \Delta_2 - (A_2 + B_2) \\
\vdots \\
R_p(\Delta_p) \xrightarrow{A_p \twoheadrightarrow B_p} \begin{matrix} R_{p1}(A_p, B_p) \\ R_{p2}(A_p, \ '_p) \end{matrix} & \text{where } \Gamma'_p = \Delta_p - (A_p + B_p)
\end{cases}
$$

and thus we get

$$\sum_{\text{for all } R_i'' \in R''} |R_i''| = |A_1| + |B_1| + |A_1| + |\Delta_1| - |A_1| - |B_1| + |A_2| + |B_2| + |A_2| + |\Delta_2| - |A_2| - |B_2| + |A_3| + |B_3| + |A_3| + |\Delta_3| - |A_3| - |B_3| \\ \vdots \\ + |A_p| + |B_p| + |A_p| - |\Delta_p| - |A_p| - |B_p|$$

$$\sum_{\text{for all } R_i'' \in R''} |R_i''| = (|\Delta_1 + \Delta_2 + \Delta_3 + \ldots + \Delta_p|) + |A_1| + |A_2| + \ldots + |A_p| \qquad (5.2.3.3)$$

$$\sum_{\text{for all } R_i'' \in R''} |R_i''| = \sum_{m=1}^{p} |\Delta_m| + \sum_{m=1}^{p} |A_m| \qquad (5.2.3.4)$$

$$\sum_{\text{for all } R_i' \in R'} |R_i''| = \sum_{m=1}^{p} |\Delta_m + A_m| \qquad (5.2.3.5)$$

Now, by analyzing formulas (5.2.3.2) and (5.2.3.4), we can conclude that choosing a common dependency (if any) for decomposition is always better (in the sense of optimality) than considering an arbitrary dependency, even if it has the

smallest left side in the set of data dependencies. Indeed, we need to prove that for all situations, the value of (5.2.3.2) is less than the value of (5.2.3.4).

Consider the formula (5.2.3.2) which calculates the number of attributes (including repetitions) of a schema (or a subschema), for a situation that all of the relation schemes of the schema are decomposed based on the same data dependency (i.e., a dependency holding in all of them). Now, assume that one of these schemes is not decomposed based on the common dependency, but it is decomposed based on an arbitrary dependency. Then, as it is shown below, there will be an increase in the number of attributes of the schema.

(a) if all $R_i \in R$ are decomposed based on the same data dependency $X_j \longrightarrow\!\!> Y_j$:

$$\sum_{\text{for all } R_i \in R} |R_i| = \sum_{m=1}^{p} |\Delta_m| + |X_j| - (p-1)\cdot|Y_j| \qquad \text{(a.1)}$$

(b) if all $R_i \in R$ (except R_k) are decomposed based on the same data dependency $X_j \longrightarrow\!\!> Y_j$, and R_k is decomposed based on an arbitrary dependency $A_k \longrightarrow B_k$:

$$\sum_{\text{for all } R_i \in R - R_k} |R_i| \quad \sum_{m=1,\, m\neq k}^{p} |\Delta m| + |X_j| - [(p-1)-1] \cdot |Y_j| \qquad \text{(b.1)}$$

$$|R_k| = |A_k| + |B_k| + |A_k| + |\Delta_k| - (|A_k| + |B_k|) = |A_k| + |\Delta_k| \qquad \text{(b.2)}$$

if we add (b.1) and (b.2) together we get

$$\sum_{\text{for all } R_i \in R} |R_i| = \sum_{m=1,\, m \neq k}^{p} |\Delta_m| + |X_j| - (p - 2) \cdot |Y_j| + |A_k| + |\Delta_k|$$

$$= \sum_{m=1}^{p} |\Delta_m| + |X_j| - (p - 2) \cdot |Y_j| + |A_k| \qquad \text{(b.3)}$$

and we conclude that, (b.3) is always greater than (a.1), by indicating that the expression [(b.3) - (a.1)] is always positive.

$$\sum_{\text{for all } R_i \in R} |R_i| \text{ of (b.3)} - \sum_{\text{for all } R_i \in R} |R_i| \text{ of (a.1)} = |A_k| + |Y_j|$$

As conclusion, the number of attributes increases at least by 2 (when $|A_k| = |Y_j| = 1$). By the same token, if the number of relation schemes that are decomposed based on arbitrary dependencies is p', then the number of attributes in the schema increases by

$$\sum_{k=1}^{p'} |A_k| + p' \cdot |Y_j|.$$

The above analysis and the Proposition 5.4, lead to the conclusion that, when there are choices for decomposition, we can construct an optimal schema if: (i) those schemes that have some data dependencies in common, are decomposed based on the common dependency with the smallest left side,

completing the proof. #

Considering Proposition 5.4 and Theorem 5.3, now it is possible to modify the FNF Algorithm to construct an optimal schema. To do so, we actually need to modify only those PARTs of the algorithm in which decompositions take place. In the FNF Algorithm, a decomposable scheme is decomposed based on a data dependency holding in it which is encountered first by the algorithm. Now, since the intention is the construction of an optimal schema, the algorithm should categorize all decomposable schemes before practically decomposing any scheme. Then considering Theorem 5.3, it should decompose each relation scheme based on an appropriate data dependency. One suggestion which leads to an efficient implementation of the problem is given in the following algorithm.

5.2.3.1. Description of the Optimal Decomposition Algorithm

For this (Figure 5-3) algorithm we need to construct a m x n matrix M, where m is the number of decomposable schemes R in the schema and n is the cardinality of the set of data dependencies G which is to be checked for decomposition. Then, $M_{i,j} = 1$, if the data dependency $g_j \in G$ holds for relation scheme $R_i \in R$ (i.e., if R_i can be decomposed based on g_j) and $M_{i,j} = 0$ otherwise. In addition we need to define an array A of n elements, so that A_k states the number of decomposable schemes that data dependency g_k holds for them.

Algorithm 5-1

An Algorithm to Decompose a Set of Decomposable Schemes into an Optimal Form

INPUT: A set R of m decomposable relation schemes; and a set G of n data dependencies.

OUTPUT: A set of decomposed schemes in optimal form.

STEP 1: Find the element of array A, say A_k, which contains the largest number (if more than one element found, then choose the one such that the corresponding dependency g_k has the smallest left side), then remove this element.

STEP 2: Decompose all relation schemes in which data dependency g_k holds. These schemes can be found simply by scanning column k of matrix M. That is, any row that has a '1' in column k, corresponds to one of these schemes. Mark these schemes to indicate that they have been decomposed. Then update array A, that is, for any $R_i \in R$ that was just marked and $M_{i,j} = 1$, decrease A_j by one.

STEP 3: Repeat STEP 1 and STEP 2 until all $R_i \in R$ are marked, that is, all decomposable schemes are decomposed.

Figure 5-3

5.3 Bernstein's Synthetic Approach

There have been several approaches for designing an algorithm to synthesize the relational schema from a set of functional relationships. Wang and Wedekind [29] proposed such an algorithm to produce third normal form relations from a given set of functional dependencies. However, their algorithm could generate two relations with keys that are functionally equivalent. Another approach has been made by Bernstein [8]. Bernstein's work was based on the approach given by Delobel and Casey [15]. The relational schema produced by his algorithm is in third normal form and contains a minimal number of relations. The fourth normal form schema cannot be achieved by this algorithm because, only functional dependencies have been considered and multivalued dependencies have been ignored. This algorithm is indicated in Figure 5-4.

Example 5-6: Given the following functional dependencies:

f_1: A $\rightarrow$ X

f_2: A,B $\rightarrow$ Y

f_3: A,B $\rightarrow$ Z

f_4: B,Z $\rightarrow$ A

f_5: B,Z $\rightarrow$ Y

f_6: B,Y $\rightarrow$ A

f_7: B,Y $\rightarrow$ Z

The nonredundant covering of the above set of FDs is as follows:

f_1: A $\rightarrow$ X

Algorithm 5-2

Bernstein's Algorithm to synthesize 3NF schema

INPUT : A set F of FDs.

OUTPUT: A set of relation schemes in 3NF.

STEP 1: Find a nonredundant covering for F. Call this set G.

STEP 2: Partition all functional dependencies in G, into groups where all of the FDs in each group have identical left sides.

STEP 3: Let G_1 and G_2 be any pair of groups, then merge these two groups together if there exists a bijection $X \longleftrightarrow Y$ (X and Y are left sides of groups G_1 and G_2 respectively) in the closure of G, G^+.

STEP 4: Construct a relation scheme for each group by taking the set union of all the attributes appearing in that group. The set of attributes that appears on the left side of any FDs of that group is the key of the relation scheme.

Figure 5-4

f_2: A,B $\longrightarrow$ Y

f_4: B,Z $\longrightarrow$ A

f_7: B,Y $\longrightarrow$ Z

Notice that f_3 is redundant because it could be derived from f_2 and f_7 using Axiom FD4. Similarly f_5 and f_6 are redundant because f_5 is implied by f_4 and f_2, and f_6 is implied by f_7 and f_4.

Now, from the above set of nonredundant covering, the algorithm constructs the following third normal form relational schema:

R_1(<u>A</u>, X)

R_2(<u>A</u>, <u>B</u>, Y)

R_3(<u>B</u>, <u>Z</u>, A)

R_4(<u>B</u>, <u>Y</u>, Z)

An important problem concerning the Algorithm 6-1 is finding the nonredundant covering of the set of FDs (STEP 1) efficiently. Recall from Chapter II that an FD $f_i \in F$ is said to be redundant in F if $F^+ = (F - \{f_i\})^+$. In other words, if $f_i \in (F-\{f_i\})^+$, then f_i is redundant and can be removed from the set F without altering the closure of the set. To find the nonredundant covering of F, we need to find the FDs that are redundant, and then remove them from the set. Therefore, an algorithm which can decide whether or not a single functional dependency is in the closure of a given set of FDs, appears to be essential for synthesizing relations from FDs. A membership algorithm is given in the appendix.

5.4 Fagin's Decomposition Approach

In [18], Fagin has presented a decomposition approach for constructing the relational schema. In this approach, he considers both kind of dependencies, functional and multivalued, and the schema constructed is in 4NF. He begins his normalization process by forming a single relation scheme consisting of all attributes of the data base. Then this relation (if not in 4NF) would be decomposed into two of its projections. This process continues for each subrelation not in 4NF, until all of them are in 4NF. The following example indicates a 4NF normalization process.

Example 5.7: Reconsider the data base of the Example 3.12 given in Chapter III. The data dependencies are as follows:

f_1: CLASS,SECTION $\rightarrow$ INSTRUCTOR

f_2: CLASS,SECTION,DAY $\rightarrow$ ROOM

f_3: STUDENT $\rightarrow$ MAJOR,YEAR

f_4: INSTRUCTOR $\rightarrow$ RANK,SALARY

m_1: CLASS,SECTION $\rightarrow\!\!\!\!\rightarrow$ STUDENT,MAJOR,EXAM,YEAR

m_2: CLASS,SECTION $\rightarrow\!\!\!\!\rightarrow$ INSTRUCTOR,RANK,SALARY

m_3: CLASS,SECTION $\rightarrow\!\!\!\!\rightarrow$ DAY,ROOM

m_4: CLASS $\rightarrow\!\!\!\!\rightarrow$ TEXT

m_5: CLASS,SECTION,STUDENT $\rightarrow\!\!\!\!\rightarrow$ EXAM

The normalization process begins by forming a single relation scheme R containing all attributes:

R(CLASS,SECTION,STUDENT,MAJOR,EXAM,YEAR,INSTRUCTOR,
RANK,SALARY,TEXT,DAY,ROOM)

Based on the MVD m_1 the relation scheme R can be decomposed into R_1 and R_2 as follows:

R_1(CLASS,SECTION,STUDENT,MAJOR,EXAM,YEAR)

R_2(CLASS,SECTION,INSTRUCTOR,RANK,SALARY,TEXT,DAY,
ROOM)

Neither R_1 nor R_2 is in 4NF. R_1 can be decomposed based on m_5 into

R_{11}(CLASS,SECTION,STUDENT,EXAM)

R_{12}(CLASS,SECTION,STUDENT,MAJOR,YEAR)

Although R_{11} is in 4NF, R_{12} is not. It can be decomposed based on FD f_3 into

R_{121}(STUDENT,MJAOR,YEAR)

R_{122}(CLASS,SECTION,STUDENT)

Both R_{121} and R_{122} are in 4NF. We now decompose R_2 based on MVD m_2 into

R_{21}(CLASS,SECTION,INSTRUCTOR,RANK,SALARY)

R_{22}(CLASS,SECTION,TEXT,DAY,ROOM)

Considering FD f_4, R_{21} can be decomposed into

R_{211}(INSTRUCTOR,RANK,SALARY)

R_{212}(CLASS,SECTION,INSTRUCTOR)

And using MVD m_4, R_{22} can be decomposed into

R_{221}(CLASS,TEXT)

R_{222}(CLASS,SECTION,DAY,ROOM)

Now, the schema consisting of the set of relation schemes $\{R_1, R_{121}, R_{122}, R_{211}, R_{212}, R_{221}, R_{22}\}$ is in 4NF. If we

remove R_{122} from the schema (because it is a projection of R_{11}, and therefore it is redundant) we get the same set of relation schemes that we got earlier in Chapter III, using the new method.

As we saw in Section 5.3 and in this section, the Bernstein's synthetic approach guarantees 3NF schema and not higher, while the decomposition process can lead to a 4NF family. In addition, synthesized schema satisfies the representation principle of Definition 5.2 given in Section 5.2, and decomposition leads to a schema which satisfies the representation principle defined by Definition 5.1 [6]. Although the schema constructed by Fagin's decomposition method is in 4NF, it is not necessarily optimal. On the other hand, as it was shown in previous chapters, the dependency-to-relation approach not only guarantees the 4NF schema, it also can lead to an optimal schema. In addition, this method does not violate the representation principle of Definition 5.1, and avoids violating the Definition 5.2 as much as possible.

Considering the above discussion and the idea given in [6], we can summarize the differences in the following table:

Method	Bernstein's	Fagin's	Authors
Normal Form	3NF	4NF	4NF
Data Dependencies	FDs	FDs+MVDs	FDs + MVDs
Definition of Minimality	Def.5.3	——	Both, Def. 5.3 and Def. 5.4
Definition of Representation	Def. 5.2	Def.5.1	Def. 5.1 and mostly Def. 5.2

5.5. Chapter Summary and Remarks

Properties of the relation schemes constructed by the FNF Algorithm have been discussed in detail. It has been proven that the schema constructed is in 4NF and thus free of undesirable properties. It has been also shown that the schema may possibly contain some redundant information, and thus it is not optimal. To solve this problem, heuristics have been employed and "optimal" has been defined. Furthermore, an algorithm (which can be efficiently implemented) has been presented for decomposition of a set of relation schemes into an optimal form. Finally, properties of relations by the FNF Algorithm and by the other approaches have been compared.

CHAPTER VI

SUMMARY AND CONCLUSION

6.1 Summary

It was the purpose of this thesis to investigate the basic concepts of the relational model, and to present an approach for constructing relational schema from functional and multivalued dependencies. Although some of the previous works [8,11,14,18, et al.] have addressed a similar problem, these approaches present a number of situations that are not desirable for the data base. Many of these problems which will be pointed out in the next section have been resolved by the new method presented in this thesis.

The theoretical background for the approach was provided in Chapter II. The algebraic rules for both kind of dependencies (functional and multivalued), were examined. We also proposed two special closures for mixed dependencies (i.e., FDs and MVDs), and demonstrated their importance for the relational schema design. Two algorithms for computation of these closures along with the extensive analyses of them were given in Chapter IV.

The main algorithm of the approach was presented in

Chapter III. The process was started with designing a simple algorithm to construct one relation scheme for each explicitly designated dependency. We then examined this simple algorithm, and discovered a number of problems presented by it. These problems which were imputed to the ignorance of the composing rules for data dependencies, led us to modify the algorithm. Finally, after an iterative refinement of the algorithm, we presented the main algorithm (Fourth Normal Form Algorithm).

In Chapter V we examined the important properties of the schema constructed by the FNF Algorithm.

We proposed (in Section 5.2) a new concept for optimality, and made an extensive investigation toward constructing optimal relational schema.

Finally, in sections 5.3 and 5.4, we examined the Bernstein's synthetic algorithm and Fagin's decomposition method. We also compared their methods with the new approach presented in this dissertation.

6.2 Analysis of the Approach

First of all, since both kind of dependencies have been considered for the design approach, the constructed schema is in 4NF, and therefore many of the problems concerning the synthesized 3NF schema have been eliminated.

In addition, allowing the data base administrator

to inspect the schema at any desirable intermediate level, leads to practical results as well as theoretical consequences. This intervention can be performed efficiently since it might be categorized based on the different parts of the FNF algorithm. In other words, the result of each part of the algorithm can be examined independently -- one part eliminates explicit partial dependencies, another part concentrates on implicit partial dependencies, another part is concerned with transitive dependencies, etc. As an example, consider a situation in which the set of dependencies designated by the data base administrator includes the following:

f_1: EMPLOYEE $\longrightarrow$ MANAGER

f_2: MANAGER $\longrightarrow$ SALARY

f_3: EMPLOYEE $\longrightarrow$ SALARY.

Syntactically, the dependency f_3 can be derived from dependencies f_1 and f_2 using Axiom FD3. Note that what actually is implied by f_1 and f_2 is an FD in which EMPLOYEE determines the SALARY of the MANAGER, and this is completely different from f_3 in which EMPLOYEE determines its own SALARY. Therefore, f_3 is not redundant and should not be removed from the schema. This problem can be overcome by considering a signal in PART-8 when implementing the FNF Algorithm. This signal warns the data base administrator about deleting a nonredundant relation from the schema, and hence he can make the proper decision

regarding this matter.

Moreover, using the concepts given in Subsection 5.2.3, the FNF Algorithm can be further modified to produce an optimal schema. This could save a large amount of storage for storing the data, because the repetition of the attributes has been minimized.

Finally, since the normalization process is performed in a step by step manner (i.e., generating 1NF schema, then transforming the 1NF into 2NF, 2NF into 3NF, etc.), the results can be used for the analysis of different normal forms and their transformations.

6.3 Suggestions for Further Work

This dissertation dealt solely with the logical issue of the relational data base. Specifically speaking, we focused more on the logical significance of the proposed algorithms than on their efficiency. Therefore, an improvement to the approach will be desirable regarding this matter.

Although this work provides all relevant algorithms, no actual programming has been done. These algorithms, especially the FNF Algorithm and those that are used for constructing the closure II and closure III, should be tested on real data bases, and their performance evaluated.

Another direction worth investigating is improving the approach to construct the relational schema satisfying

the representation principle of Definition 5.1, and not violating the Definition 5.2.

Finally, this dissertation leaves the designating of functional and multivalued dependencies completely in the hands of the data base administrator. Some work is needed for the systematic detection of these dependencies.

BIBLIOGRAPHY

1. Aho, A.V., C. Beeri, and J.D. Ullman, "The theory of joins in relational databases", ACM Transactions on database Systems, September 1979.

2. Aho, A.V., J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison Wesley, Mass., 1974.

3. Armstrong, W.W., "Dependency structure of data base relationships", Proc. 74 IFIP, North-Holland, 1974.

4. Beeri, C., "On the membership problem for multivalued dependencies in relational databases", TR229, Dept. of EECS, Princeton University, 1977.

5. Beeri, C. and P.A. Bernstein, "Computational problems related to the design of normal farm relation schemes", ACM Transactions on Database Systems, 1979.

6. Beeri, C., P.A. Bernstein, and N. Goodman, "A sophisticate's introduction to database normalization theory", Proc. ACM Intl. Conf. on very large Data Bases, 1978.

7. Beeri, C., R. Fagin, and J.H. Howard, "A complete axiomatization for functional and multivalued dependencies", ACM/SIGMOD International Symposium on Management of Data, 1977.

8. Bernstein, P.A. "Synthesizing third normal form relations from functional dependencies", ACM Trans. on Database Systems. 1976.

9. Cadiou, J.M. "On Semantic Issues in the Relational Model of Data", IBM Research Laboratory, San Jose, CA, 1977.

10. Codd, E.F. "A relational model for large shared data banks", CACM, 1970.

11. Codd, E.F. "Further normalizations of the data base relational model", In Data Base Systems, Courant Inst. Computer Science Symposium 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, NJ, 1972.

12. Codd, E.F. "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, NJ, Prentice-Hall, 1972.

13. Data, C.J. "An Introduction to Database Systems, Addison Wesley, Mass., 1977.

14. Delobel, C., "Normalization and hierarchical dependencies in the relational data model", ACM Transactions on Database Systems, 1978.

15. Delobel, C. and R.C. Casey, "Decomposition of a database and the theory of Boolean Switching functions", IBM J. Res., 1972.

16. Fagin, R. "Functional dependencies in a relational database and propositional logic", IBM Res. Laboratory, San Jose, CA, 1976.

17. Fagin, R. "The decomposition versus synthetic approach to relational database design. Proc. Third Intl. Conf. on very large data bases, Tokyo, October 1977.

18. Fagin, R., Multivalued dependencies and a new norman form for relational databases", ACM Trans. on Database Systems, 1977.

19. Fagin, R., "Functional dependencies in a relational database and propositional logic", IBM J. Res. and Develop., November 1977.

20. Fadious, R., "Mathematical foundations for relational data bases", doctoral diss., Michigan State Univ., 1975.

21. Hutt, A.T.F., "A relational data base management system", John Wiley & Sons, NY 1979.

22. Hagihard, K., M. Ito, K. Tanighuchi and T. Kasami, "Decision problems for multivalued dependencies in relational databases", SIAM J. Comput., May 1979.

23. Luk, W.S., "Possible membership of a multivalued dependency in a relational database", Information Processing letters, August 1979.

24. Meldelzon, A.O. "On axiomatizing multivalued dependencies in relational databases" J. ACM, 1979.

25. Martin, J., "Computer data-base organization", Prentice-Hall, Englewood Cliffs, NJ 1975.

26. Rissanen, J. "Independent components of relations", ACM Trans. on Database Systems, 1977.

27. Schmid, H.A. and J.R. Swenson, "On the semantics of the relational model", ACM/SIGMOD International Symposium on Management of Data, 1976.

28. Stant, D.F., and D.F. McAllister, "Discrete Mathematics in Computer Science", Prentice-Hall, 1977.

29. Wang, C.P. and H.H. Wedekind, "Segment Synthetics in Logical Data Base Design", IBM J. of Res. and Dev., January 1975.

30. Wiederhold, G. "Data base design", McGraw-Hill, NY, 1977.

31. Zaniolo, C. "Analysis and design of relational schemata for database systems", doctoral diss., UCLA, 1976.

APPENDIX

Completeness of the inference rules for FDs and MVDs[†]

Using the definition it is easy to verify the validity of the inference rules. To prove the completeness of this set of rules we need to show that each dependency which is implied by $F \bigcup M$ (F is a set of FDs, and M is a set of MVDs) belongs to $(F,M)^+$.

Recall that a dependency f is implied by $F \bigcup M$ if there is no counterexample relation such that all dependencies in $F \bigcup M$ are valid in it but f is not. To show completeness of the rules, we have to show that for each dependency not in $(F,M)^+$ such a counterexample relation does exist. Before we present the completeness theorem we need a few concepts.

Let X be a subset of U. There are several sets Y such that the MVD $X \twoheadrightarrow Y$ is in $(F,M)^+$, (e.g., $X \twoheadrightarrow U-X$ is always in $(F,M)^+$. Following Fagin, we use the notation $X \twoheadrightarrow Y_1|Y_2|\ldots|Y_k$ to denote the collection of MVD's $X \twoheadrightarrow Y_1$, $X \twoheadrightarrow Y_2, \ldots, X \twoheadrightarrow Y_k$. From now on,

---

[†]This discussion follows quite closely that of Beeri, Fagin and Howard [7].

when this notation is used, we assume that none of the sets $Y_1,\ldots,Y_k$ is the empty set.

Let us denote by DEP(X) the family of all sets Y for which $X \twoheadrightarrow Y$. (DEP(X) is, of course, a function of the given sets of dependencies F and M.) We have seen that DEP(X) is closed under boolean operations (MVD5,MVD6). Therefore, it contains a unique subfamily with the following properties:

a) The sets in the subfamily are nonempty.

b) Each pair of sets in the subfamily is disjoint.

c) Each set in DEP(X) is a union of sets from the subfamily.

This subfamily consists of all nonempty minimal sets in DEP(X), i.e., those sets that do not contain any other nonempty set of DEP(X). We call this subfamily the <u>dependency basis</u> of X. If $Y_1,\ldots,Y_k$ are the sets in the dependency basis of X, then as Fagin noted [18], "the generalized" MVD $X \twoheadrightarrow Y_1|\ldots|Y_k$ contains all the information about MVDs that have X as their left side.

Let X* denote the set of all attributes that are functionally dependent on X (by functional dependencies in $(F,M)^+$). Clearly $X \subseteq X^*$. X* has the same role for FD's as DEP(X) has for MVD's. For each $A \in X^*$ we have $X \rightarrow A$. Thus, each element of X* appears as a singleton set in the dependency basis of X. The dependency basis contains other sets if and only if X* is a proper subset of U.

These remaining sets cover U-X*. We note that since $X \rightarrow X^*$ and $X^* \rightarrow X$, it follows that DEP(X) = DEP(X*) and the dependency basis of X is the same as the dependency basis of X*.

Theorem 1. (Completeness theorem for FDs and MVDs): Let F and M be sets of FDs and MVDs, respectively. For each functional or multivalued dependency that does not belong to $(F,M)^+$ there exists a relation R(U) such that all the dependencies in $(F,M)^+$ are valid in R but the given dependency is not valid in R.

Proof: Let X be the left side of the dependency that is not in $(F,M)^+$. The set X* is a proper subset of U since otherwise every (functional or multivalued) dependency with left side X belongs to $(F,M)^+$. Let $W_1,\ldots,W_m$ $(m \geq 1)$ be the sets in the dependency basis of X that cover U-X*. Thus, X*, $W_1,\ldots,W_m$ form a partition of U. The MVD $X \rightarrow\!\!\!\rightarrow X^* W_1 \ldots W_m$ is in $(F,M)^+$ and, furthermore, if an MVD in $(F,M)^+$ has X as its left side then its right side is a union of a subset of X* and some of the sets $W_1,\ldots,W_m$.

The relation R(U) is constructed as follows: We choose the set {0,1} as the domain of each of the attributes in U. The relation R has 2^m rows, one row for each sequence of zeros and ones of length m. In the row corresponding to a sequence $\langle a_1,\ldots,a_m\rangle$ (where $a_i \in \{0,1\}$), each of the attributes in W_i is assigned the value a_i $(i = 1,\ldots,m)$. Each attribute in X* is assigned the value

1 in all the rows of the relation. For example, if m = 3, then the row corresponding to the sequence {0,1,1} has all 1's in the X* columns, all 0's in the W_1 columns, all 1's in the W_2 columns and all 1's in the W_3 columns.

We now want to prove that R satisfies the condition of the theorem. In what follows, we use the inference rules for two different purposes. First, we sometimes show that if some given dependencies are in $(F,M)^+$ then $(F,M)^+$ also contains some other dependency. That we can use the rules for this purpose follows directly from the definition of $(F,M)^+$. Second, we also show that if some dependencies are valid in R then there is another dependency that is valid in R. We can use the rules for this purpose since we have proved that they are valid inference rules for the family of dependencies, i.e., their application to dependencies that are valid in a relation always produces dependencies valid in that relation. We will indicate our intention each time we use the rules.

We now prove the following three claims about the relation R that we have just constructed.

(1) If the right side of an FD is a nonempty subset of W_i then the FD is valid in R iff its left side intersects W_i.

(2) Each MVD that has W_i as its right side is valid in R.

(3) If the right side of an MVD is a nonempty proper subset of W_i then the MVD is valid in R iff its left

side intersects W_i.

We first prove one direction of claim 1. For each fixed row of R, all the attributes in W_i have the same value. It follows that every attribute in W_i is functionally dependent in R on every other attribute in W_i (and, by augmentation, on every set that contains such an attribute). Thus we have proved that if the left side of the FD intersects W_i then the FD is valid in R. From this also follows the corresponding direction of claim 3, since every FD is also an MVD.

We now prove claim 2 and the other directions of claims 1 and 3. The relation R is the Cartesian product of its projections $R(W_i)$ and $R(U-W_i)$. It follows immediately that the MVD $\emptyset \rightarrow\!\!\!\!\rightarrow W_i$ is valid in R and, by augmentation, $Y \rightarrow\!\!\!\!\rightarrow W_i$ is valid in R, for every set Y. This proves claim 2. Now let Y and Z be subsets such that Y is disjoint from W_i and Z is a nonempty subset of W_i. It follows from the above factorization of R that for each Y-value (Def: For a set of attributes X, an X-value is an assignment of values to the attributes of X from their domains) y, the set $Z_r(y)$ contains two Z-values - a 0-assignment and a 1-assignment to the attributes of Z. In particular, the FD $Y \rightarrow Z$ is not valid in R; this concludes the proof of claim 1. If Z is a proper subset of W_i let A be an attribute in $W_i - Z$. Then $Z_r(ya)$, where ya is a YA-value, contains only a single Z-value since the

attributes of Z must be assigned the same value as the attribute A. Therefore $Z_R(y) \neq Z_R(ya)$ and it follows that the MVD $Y \twoheadrightarrow Z$ is not valid in R. This concludes the proof of claim 3.

We now show that R satisfies the condition of the theorem. First, let f be an FD in $(F,M)^+$. We show that f is valid in R. By Axiom FD6 we can assume that f is of the form $Y \rightarrow B$ where B is a single attribute. Now, if B is in X* then f is clearly valid in R since in R every attribute of X* assumes a single value and is, therefore, functionally dependent on any other attribute. If B is not in X* then it belongs to some W_i. If Y is disjoint from this W_i then from the MVD $X \twoheadrightarrow W_i$ and the FD $Y \rightarrow B$ which are both in $(F,M)^+$ it would follow by rule FD-MVD2 that $X \rightarrow B$ is in $(F,M)^+$. This is impossible since B is not in X*. Therefore Y must intersect W_i and $Y \rightarrow B$ is valid in R by claim 1.

Now let g: $Y \twoheadrightarrow Z$ be an MVD in $(F,M)^+$. We show that g is valid in R. We note that $Y \rightarrow Z \cap X^*$ is valid in R. We will show that, for each i, $Y \twoheadrightarrow Z \cap W_i$ is also valid in R. First, suppose that, for some i, the set $Z \cap W_i$ is either empty or all of W_i. By claim 2 above, $Y \twoheadrightarrow W_i$ is valid in R; as we know, $Y \twoheadrightarrow \emptyset$ is always valid. Next, suppose that, for some i, $Z \cap W_i$ is a non-empty proper subset of W_i. If Y does not intersect W_i we can use augmentation on $Y \rightarrow Z$ to obtain that $U-W_i \twoheadrightarrow Z$

is in $(F,M)^+$. Since $X \twoheadrightarrow W_i$ is also in $(F,M)^+$ it follows by MVD3 that $X \twoheadrightarrow Z-(U-W_i)$, that is $X \twoheadrightarrow Z \cap W_i$ is in $(F,M)^+$. This is a contradiction to the assumption that W_i is a member of the dependency basis of X. Thus Y must intersect W_i and, by claim 3, $Y \twoheadrightarrow Z \cap W_i$ is valid in R. We have now shown that, for each i, $Y \twoheadrightarrow Z \cap W_i$ is valid in R and also so is $Y \twoheadrightarrow Z \cap X^*$. By taking the union (MVD4) it follows that $Y \twoheadrightarrow Z$ is valid in R.

Finally, let us consider the dependency (with left side X) which is known not to be in $(F,M)^+$. If it is an FD $X \rightarrow Y$ then Y is not a subset of X^*, so Y intersects W_i for some i. By FD6 if $X \rightarrow Y$ is valid in R so is $X \rightarrow Y \cap W_i$ and this contradicts claim 1. (Recall that X^* is disjoint from each of the W_i.) Therefore $X \rightarrow Y$ is not valid in R. If the dependency is an MVD $X \twoheadrightarrow Y$, then for some i, $Y \cap W_i$ must be a nonempty proper subset of W_i (else since $C \rightarrow Y \cap X^*$ and $X \twoheadrightarrow Y \cap W_i$ for each i are in $(F,M)^+$ the MVD $X \twoheadrightarrow Y$ would be in $(F,M)^+$). Now, for this i, $X \twoheadrightarrow Y \cap W_i$ is not valid in R by claim 3. Since $X \twoheadrightarrow W_i$ is valid in R, if $X \twoheadrightarrow Y$ was also valid in R we could apply MVD6 to obtain a contradiction. Thus $X \twoheadrightarrow Y$ is not valid in R. #

Mendelzon's Proof of Proposition 2.2[†]

Proposition. The Union rule (i.e., MVD5) cannot be derived from rules MVD1-MVD3.

Proof: Let U = {A,B,C}, G = {A $\twoheadrightarrow$ B, A $\twoheadrightarrow$ C}, and g = A $\twoheadrightarrow$ BC. Clearly g follows from G via MVD5. Let S = {MVD1,MVD2,MVD3}; we claim that $G^S \subseteq G'$, where G^S is the closure of G under S, and G' = {X $\twoheadrightarrow$ Y/Y $\subseteq$ X or A $\in$ X and Y-X = B or C}. To prove this, it suffices to show that G' is closed under S, since clearly G is included in G'. We shall consider each rule of S in turn.

MVD1. Obviously G' contains all dependencies obtainable by reflexivity.

MVD2. Suppose X $\twoheadrightarrow$ Y $\in$ G' and X $\subseteq$ W. Consider two cases.

(a) Y $\subseteq$ X; then YZ $\subseteq$ XW implies that XW $\twoheadrightarrow$ YZ $\in$ G'.

(b) Y $\subseteq$ X, A $\in$ X, and Y-X = B or C. Suppose Y-X = B. Then YZ-XW=(Y-X-W)(Z-X-W) = B-W. If B $\in$ W, then YZ-XW=$\emptyset$, which implies that YZ is contained in XW, so XW $\twoheadrightarrow$ YZ $\in$ G'. If B $\notin$ W, then YZ-XW = B and A $\in$ XW implies that XW $\twoheadrightarrow$ YZ $\in$ G'. A similar argument holds for Y-X = C.

MVD3. Suppose X $\twoheadrightarrow$ Y, Y $\twoheadrightarrow$ Z $\in$ G'. There are

---

[†]This proof follows quite closely that of Mendelzon [24].

four cases to consider.

(a) $Y \subseteq X$, $Z \subseteq Y$; then X contains Z-Y, so $X \twoheadrightarrow Z-Y \in G'$.

(b) $Y \subseteq X$, Y does not contain Z, and $A \in Y$, $Z-Y = B$ or C.

Assume $Z-Y = B$. If $B \in X$, then $Z-Y-X=\emptyset$ implies that X contains Z-Y, hence $X \twoheadrightarrow Z-Y \in G'$. If $B \notin X$, then $Z-Y-X=B$ and $A \in X$ (since $A \in Y$ and $Y \subseteq X$); hence $X \twoheadrightarrow Z-Y \in G'$. The argument is similar for $Z-Y = C$.

(c) $Y \not\subseteq X$, $A \in X$, $Y-X=B$ or C; $Z \not\subseteq Y$, $A \in Y$, $Z-Y = B$ or C.

Suppose $Z-Y = B$. If $B \in X$, $Z-Y-X = \emptyset$, hence $X \twoheadrightarrow Z-Y$ as above. If $B \notin X$, $Z-Y-X=B$ and we assumed $A \in X$, so $X \twoheadrightarrow Z-Y \in G'$. The argument for Z-Y is similar.

(d) $Y \not\subseteq X$, $A \in X$, $Y-X=B$ or C, and $Z \subseteq Y$. Now $Z-Y = \emptyset$, so $X \longrightarrow Z-Y \in G'$.

It follows that $A \twoheadrightarrow BC$ is not in G^S, and therefore $A \twoheadrightarrow BC$ cannot be derived from $\{A \twoheadrightarrow B, A \twoheadrightarrow C\}$ using only MVD1-MVD3. #

A Membership Algorithm[†]

A membership algorithm which can be applied well to real world applications is presented below. In this procedure EQN is a set of numbers associated with FDs in F.

Membership Algorithm

An Algorithm to decide if an FD is in the closure of a set of FDs

INPUT : A set of FDs F in canonical form; and an FD g, whose left and right sides are LM and RM, where |RM| = 1.

OUTPUT: "YES", if $g \in F^+$, and "NO" otherwise.

```
X = ∅;
procedure MEMBER(LM,RM,EQN)
do for all i ∈ EQN;
   if RIGHT_fi = RM  then
      if LEFT_fi ⊆ LM  then return('YES');
      else
         [do for all d ∈(LEFT_fi - LM);
            if d ∈X  then go to end d-loop;
            SUB ← MEMBER(LM,d,EQN - {i});
            if SUB = 'NO' then
               [EQN = EQN - {i};
                exit;
               ]
```

[†]This work was done by Dr. John C. Thompson and the author, and was presented at the Sixth Annual Computer Science Conference at Detroit, Michigan in February, 1978.

```
            end d-loop;
            if SUB = 'YES' then [X = X ⋃ d; return('YES')];
          ]
    end i-loop;
    return('NO');
    end MEMBER;
```

Proof of termination

Theorem 2 The membership algorithm halts.

Proof: The outer loop is finite because EQN is a finite set. At each iteration of the inner loop, either the value of SUB is 'YES' for all d's (including those created by recursion) in which case the algorithm terminates, or at least one FD is crossed off the set EQN. Now, since the number of elements in EQN is finite, the algorithm ultimately halts. #

Proof of correctness

Theorem 3 Upon termination of the membership algorithm, the output is "YES" if and only if $g \in F^+$.

Proof: At the first call of the procedure, the right sides of all FDs in F are checked until an FD f_i whose right side is the same as the right side of g is found (if no such an FD is found, g cannot be in the closure of F, and the algorithm correctly returns "NO" as its

output). If the left side of f_i is a subset of the left side of g, then clearly g is in the closure of F, and this is performed by the third statement of the procedure. If the left side of the f_i is not a subset of the left side of g, then the procedure is recursively called for each element d in the set difference of these two left sides (i.e., $LEFT_{f_i} - LM$). Therefore, the subsequent procedure call is similar to the first call, and it determines if there is any FD $\varepsilon(F - \{f_i\})$ with the right side of d. If found, again its left side is checked and the same procedure is continued until we get to a point in which all d's are replaced by the left sides of their corresponding dependencies (or the subsequent left sides), and then this set is either a subset of the left side of g, or it is not. The former case indicates that $g \varepsilon F^+$, and the inner loop of the algorithm does this job correctly. On the other hand, the latter condition states that f_i cannot be used to derive g from the set F, and thus the algorithm checks for the other FDs in F. Finally, if all FDs of F are checked (when the outer loop is exhausted) with the same situation as mentioned above, the algorithm returns "NO" and terminates. #

Speed analysis

The time analysis of the algorithm can be better explained by considering the worst situation that may

occur. Suppose there are n FDs as follows:

$$f_1: \ A_1, A_2, \ldots, A_{m1} \rightarrow B_1$$
$$f_2: \ A_1, A_2, \ldots, A_{m2} \rightarrow B_2$$
$$\vdots$$
$$f_n: \ A_1, A_2, \ldots, A_{mn} \rightarrow B_n$$

and suppose we want to know if an FD $\alpha \rightarrow \beta$ is in the closure of the above set. The algorithm first checks for β in the left side of FDs. Suppose there is an f_i in the set such that $B_i = \beta$ (if no such an FD exists, the algorithm terminates in $0(n)$). Now, the worst situation happens if $LEFT_{f_i} \cap \alpha = \emptyset$, and the algorithm has to search for all $A_k \in LEFT_{f_i}$, that is, for m_i attributes in $(n-1)$ dependencies. Notice that, even if $m_i > (n-1)$, the inner loop may not be executed for more than $(n-1)$ times. This is because the nth element of the set $LEFT_{f_i}$ cannot be found in the right side of the given FDs (assuming the first $(n-1)$ elements have already been found), causing an exit from the loop. Now the worst situation occurs if the first element can be found after searching the whole set, i.e., $(n-1)$ FDs. Since this dependency is removed from the set, and the element found is added to the set X (i.e., a set which keeps track of these attributes to avoid re-searching for them in the subsequent procedure calls), then for the next phase (i.e., searching for the attributes on the left side of the dependency just

removed) we have (n - 2) dependencies to search. Continuing a similar procedure we get a total of (n - 1)+(n - 2)+ ... + 1, and hence the overall time for the algorithm is below $0(n^2)$.

As the reader will notice, the algorithm first checks the right side of the FDs. And since all FDs are in canonical form (i.e., they have only one attribute on their left sides), this test can be efficiently implemented. The algorithm also checks the left side of the dependencies, but this test has been minimized. In other words, the left side of a dependency is checked only if its right side is appropriate (in the sense that this dependency might be used for derivation of the FD of interest).

In addition, in this algorithm, only those FDs that might be used for the derivation purpose will be tested, and the other dependencies will not be checked at all.