DESIGN OF RELATIONAL DATABASE SCHEMAS: THE TRADITIONAL DEPENDENCIES ARE NOT ENOUGH

by

ADEGBEMIGA OLA

B.Sc., The University of Ibadan, 1977

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1982

Department of COMPUTER SCIENCE

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date April 1982

Abstract

Hitherto, most relational database design methods are based on functional dependencies (FDs) and multivalued dependencies (MVDs). _Full mappings_ are proposed as an alternative to FDs and MVDs. A mapping between any two sets, apart from being one-one, many-one, or many-many, is either total or partial on the source and target sets. An 'into' mapping on a set, expresses the fact that an element in the set may not be involved in the mapping. An 'onto' mapping on a set is total on the set. A many-many (into,onto) mapping from set A to set B is written as

$A^{i} \; m\text{----}n \; B^{o}$.

The mappings incorporate more semantic information into data dependency specification. It is shown, informally, that the full mappings are more expressive than FDs and MVDs. Transformation rules, to generate Boyce-Codd normal form and projection-join normal form schemas from the full mappings, are defined. The full mapping/transformation rules provide a discipline for modeling nonfunctional relationships, within a synthetic approach.

Table of Contents

List of Figures

Acknowledgements

I am grateful to my supervisor Paul Gilmore for the initial ideas that lead to this thesis. His careful review of the research reports was of great value. I sincerely thank Bob Goldstein for his thorough review of the thesis. I also wish to thank Julie Shamper for reading the final document.

CHAPTER I . Introduction

This thesis deals mainly with automatic design of relational database schemas. Most design approaches hitherto have been based on the functional and multivalued dependencies. The dependencies serve as a means of expressing relationships and constraints to be observed within the database. A new method for specifying constraints and relationships, from which relation schemas can be derived, is proposed.

1.1 General Area of Research

Database design can be subdivided into three major levels.[1]

1.) the choice of database models.

2.) the design of logical database schemas.

3.) the physical database design and loading.

The logical design can be broken down further into

1.) Requirement analysis and conceptual schema design.

2.) Data model-specific schema design.

Database Models provide ways in which data are arranged and manipulated.

---

[1](Haseman and So 1977) identified these three levels.
i) choice of database models.
ii) the design of database schemas.
iii) the loading of the physical database.

(Haseman and So 1977) identified three distinct levels of database reflection of the reality. The real world, the conceptual (or information) model and the data model levels. We can conceive of a modeling continuum as in figure 1.

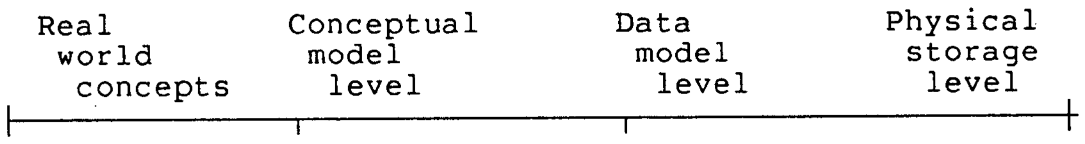

Figure 1. Modeling Continuum[2]

Other views on levels of abstraction in database design have been presented (Ullman 1980 and Date 1980). While the distinction between conceptual model and data model is explicit in (Haseman and So 1977), it is not the case in (Ullman 1980 and Date 1980).

This thesis deals with the design of relational database schemas. Following the introduction of the Relational Data Model (Codd 1970), a lot of work has been done on design and analysis of relational databases. For some time, two competing approaches have been the third normal form decomposition (Codd 1971) and the synthesis of Bernstein and others (Bernstein et al 1975 and Bernstein 1976). In (Fagin 1977a), the fourth normal form decomposition was introduced. It takes as input attributes and semantic information in the form of functional and multivalued dependencies. Recently, a number of improvements and other approaches have been proposed (Ling et al 1981, Wong et al 1980 and Zaniolo & Melkanoff 1981).

[2]It originally appeared in (Haseman and So 1977)

1.2 Proposed work

On the modeling continuum (figure 1), our design considerations lie between the conceptual model and the physical database. The input to the design process consists of Semantic and Use information, as depicted in figure 2.

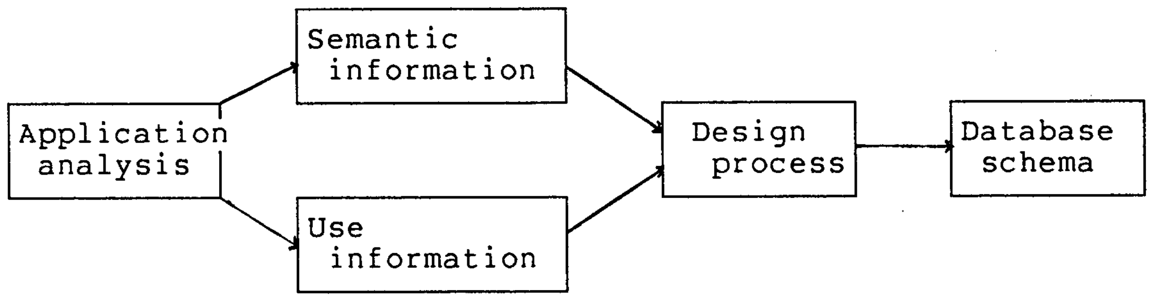

Figure 2. Schema Design[3]

Our design process consists of mapping specifications and a method for transforming the mappings into relation schemas. A mapping from one set to another represents a relationship between any two sets of objects (be it physical, abstract or structured). While the semantic information models the "real world", the use information is provided mainly to guide in the mapping specification. The semantic information reveals the inherent facts about the application environment. The use information, on the other hand, provides the use characteristics of the database, such as query types and statistics. The use information helps determine some relationships that may not be explicit from the inherent facts. In this thesis, both constraint specification and schema design are emphasized.

[3]The diagram originally appeared in (Haseman and So 1977)

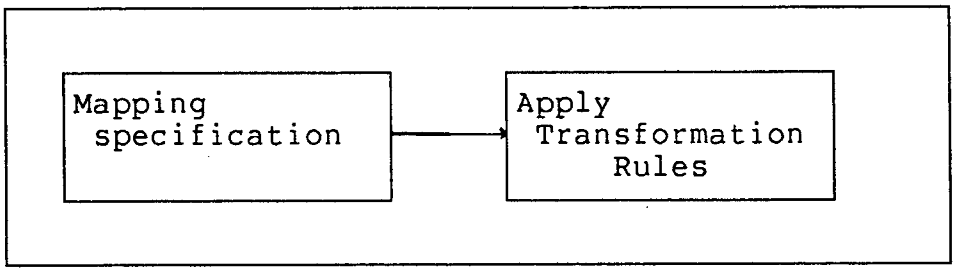

Figure 3. The Design Process

Constraints and relationships are represented by mapping types as an alternative to functional and multivalued dependencies. Subsequently, the transformation rules will be proposed. The rules generate relation schemas that conform to the Boyce-Codd and Projection-Join normal forms.

1.3 Reading Guide

The thesis is organized as follows. In chapter (2), a review of some concepts in relational database theory, concepts in information analysis, and a summary of relational database design methods are presented. The Full mapping approach to relational database design is presented in chapter (3). Full mappings, as a means of specifying relationships between data items, are proposed as an alternative to functional and multivalued dependencies. Transformation rules are formulated to generate relation schemas from full mapping specifications.
Chapter (4) is an evaluation of our theory. The effect of full mappings on the normalization process is examined. The full mapping approach is also compared with other design methods. A design example is presented in section 4.3.
Chapter (5) concludes the thesis. It summarizes our work, as

well as highlights areas that need further research.

A reader who is conversant with the relational database theory may skip most parts of chapter (2). Section 2.1 is, however, required to understand the notations and some concepts used in the thesis. A reader without much knowledge of the normalization theory may find it useful to read the section on "Normalization Process" (section 4.2) before chapters (2) and (3).

CHAPTER II . Relational Database Design: Framework

The general framework for relational database design is presented in this chapter. In section 2.1 a review of some basic concepts in relational database theory and information analysis are presented, in section 2.2, a summary of current design methodologies; and in section 2.3 appraisals of the design approaches are given.

2.1 Basic Concepts and Definitions

There are various notations used in relational database literature. In this thesis the following will be used. Letters A,B,C,... denote single attributes, and W,X,Y,... sets of attributes. If X and Y are sets of attributes (not necessarily disjoint), the union of the two sets is written as XY. The projection of a relation R over the set of attributes X is represented by R[X], and the natural join of two relations R(X) and R(Y) is written as R(X).R(Y). Explanatory definitions of relevant terms and concepts are presented in the following subsections. Keywords are underlined.

2.1.1 Information Analysis

Information analysis serves as a prelude to the design process. However, most works on relational database design overlook this step. The main method of analysis in this thesis is to identify the semantic elements similar to those in Chen's Entity-Relationship model (Chen 1976). The <u>entity sets</u> that are of interest in the application environment are identified. An entity set is a collection of objects of the same "type" that we wish to record information about in the database. An object can be physical or abstract.

The type of an object is not absolute, in the sense that it can belong to more than one set at different levels of abstraction. In (Smith and Smith 1977), the notion of generalization was introduced as an abstraction which enables a class of objects to be thought of generically as a single named object. A generic hierarchy can also be defined on a set, such that each level consists of objects that share common properties. For example, an EMPLOYEE set has common properties such as (employee-number, name, age and employee-type). Its lower generic sets could be TRUCKERS, SECRETARIES and ENGINEERS, each of which we wish to record different additional information about. (Shipman 1980) also expresses similar ideas about entity types. Subtypes are defined based on the roles which certain members of a parent entity set perform.

Other semantic elements that are of interest are the <u>property-value sets</u> and the <u>structured entity sets.</u> A structured entity set represents an association among two or more other entity sets. We refer to the relationships among entity sets as <u>entity set associations.</u> A set describing an association among n entity sets is a subset of the cartesian product of the sets. This is an abstraction in which associations among entity sets are regarded as higher level objects, which can have properties defined on them (Smith and Smith 1977). Properties of entity sets are expressed as sets of attribute-value pairs. The collection of values (that are semantically possible) of an attribute form a property-value set. Thus an attribute is essentially a function which maps an entity set to a property-value set. We refer to such functions as value associations. Hence, at different levels of abstraction, entity sets will be defined, named and their intensions clearly stated.

An intension is supposed to give the meaning of a defined set or association. It helps to differentiate between abstraction levels and to resolve ambiguities that might arise from naming. Sets in a generic hierarchy necessarily have the same underlying domains; this should be obvious from their intensions. The sets, the associations and the statement of intensions, can be maintained as a database dictionary. Such a dictionary will not only serve as a basis for database design, but can be used in query processing. But dictionary implementations in the form of a supporting system will not be suitable. Hitherto, in relational database systems, there has been no provision for stating the intensions of attributes and the dependencies among them. We suggest an approach whereby the dictionary information is maintained by the Database Management System in use, but the idea is not pursued further in this thesis. Database kernel and self-referential database is a wide area of research on its own.

2.1.2 Concepts in Relational Database Theory

A relation on the set of attributes {A1,A2,...,An} is a subset of the cartesian product Dom(A1) x Dom(A2) x...x Dom(An) where Dom(Ai)'s are the respective domains of the attributes. The elements of the relation are called tuples. In a database, a number of restrictions can be placed on a relation. These restrictions are expressed in a <u>relation</u> <u>schema.</u> A relation schema is a set of attributes along with a set of <u>contraints</u> (Cadiou 1975 and Fagin 1981). A relation R is said to be a valid instance of a schema R* if it has the same attributes as the schema and obeys every constraint of the schema. A property holds for a relation schema if it holds for all instances of it.

A constraint in relation schemas can be the specification of a key for the relation, a functional dependency, a multivalued dependency or a join dependency. Following (Bernstein 1976), <u>keys</u> and <u>superkeys</u> are defined as follows. Let R be a relation and let X be a subset of attributes of R. Then, X is a key of R if every attribute of R that is not in X is functionally dependent upon X and no subset of X has this property. The X-values of R determine tuples of R uniquely. A <u>superkey</u> of R is any set of attributes of R that contains a key of R. Thus every key is also a superkey. A relation may have several keys, referred to as candidate keys. One of the keys is usually chosen as the primary key and by convention, underlined in the relation schema. The set of candidate keys of R will be represented by Key(R).

<u>Functional</u> <u>Dependency</u> expresses a constraint that holds between two sets of attributes of a relation. Given a relation R, a combination of attributes Y of R is functionally dependent on attributes X of R if and only if each X-value in R has associated with it precisely one Y-value in R at any database instance. The functional dependency of Y on X, usually written as X--->Y, can be depicted by the following mapping diagram from

a set of X-value elements of a relation to the set of Y-value elements of the relation. An X-value can appear in more than one

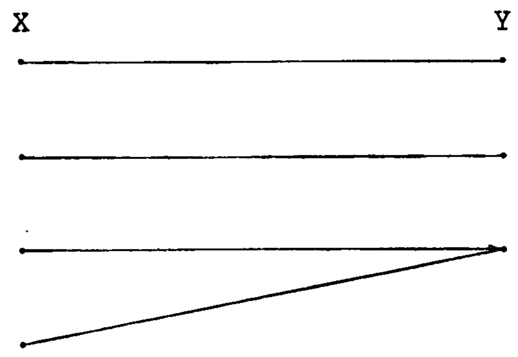

Figure 4. A Mapping Diagram for Functional Dependencies

tuple, but whenever two tuples agree on their X-values, their Y-values must also be the same. Let X and Y be combinations of attributes of R. Y is said to be <u>fully functionally dependent</u> on X if it is functionally dependent on X and not functionally dependent on any strict subset of X. The above definition of functionally dependency due to (Date 1980), is valid within the context of a relation. (Bernstein 1976) defines functional dependency between two attributes A and B as a time-varying function f:Dom(A)--->Dom(B). If f is thought of as a set of ordered pairs {(a,b): a ϵ Dom(A) and b ϵ Dom(B)}, then at any point in time, for a given value a ϵ Dom(A) there will be at most one value b ϵ Dom(B). Functional dependency and multivalued dependency are usually abbreviated as FD and MVD respectively.

<u>Multivalued Dependency</u> is defined as follows. Given a relation schema R(XYZ), the multivalued dependency of Y on X, usually written as X-->-->Y, holds in R if and only if the set of Y-values matching a given (X-value,Z-value) pair in R depends on the X-value and is independent of the Z-value. MVD can be

illustrated by the following relation which originally appeared in (Zaniolo and Melkanoff 1981).

SUPPLIER	ITEM	COLOR
WOODMAN	TABLE	BROWN
WOODMAN	SOFA	BLACK
HOUSEMAN	CARPET	RED
HOUSEMAN	CARPET	YELLOW
HOUSEMAN	CARPET	BLUE
HOUSEMAN	SOFA	BLACK
BLAND	CARPET	RED
BLAND	CARPET	YELLOW
BLAND	CARPET	BLUE

Figure 5. Sample Relation for STOCK(<u>SUPPLIER,</u> <u>ITEM,</u> <u>COLOR</u>)

In the sample relation STOCK, there is a multivalued dependency of COLOR on ITEM. Every pair of (SUPPLIER,ITEM) where item is CARPET implies that the particular supplier supplies the three colors (RED,YELLOW,BLUE). The dependency does not hold if it is possible to have a sample database as in figure 5, but with the last tuple deleted. In such a case, BLAND a supplier of carpet will supply only RED and YELLOW. From the mapping diagrams in figure 6, it is clear that multivalued dependency of COLOR on ITEM is another way of stating the fact that the relation schema STOCK is equal to the join of its projections on (SUPPLIER,ITEM) and (ITEM,COLOR). A MVD X-->-->Y in a relation R(W) is said to

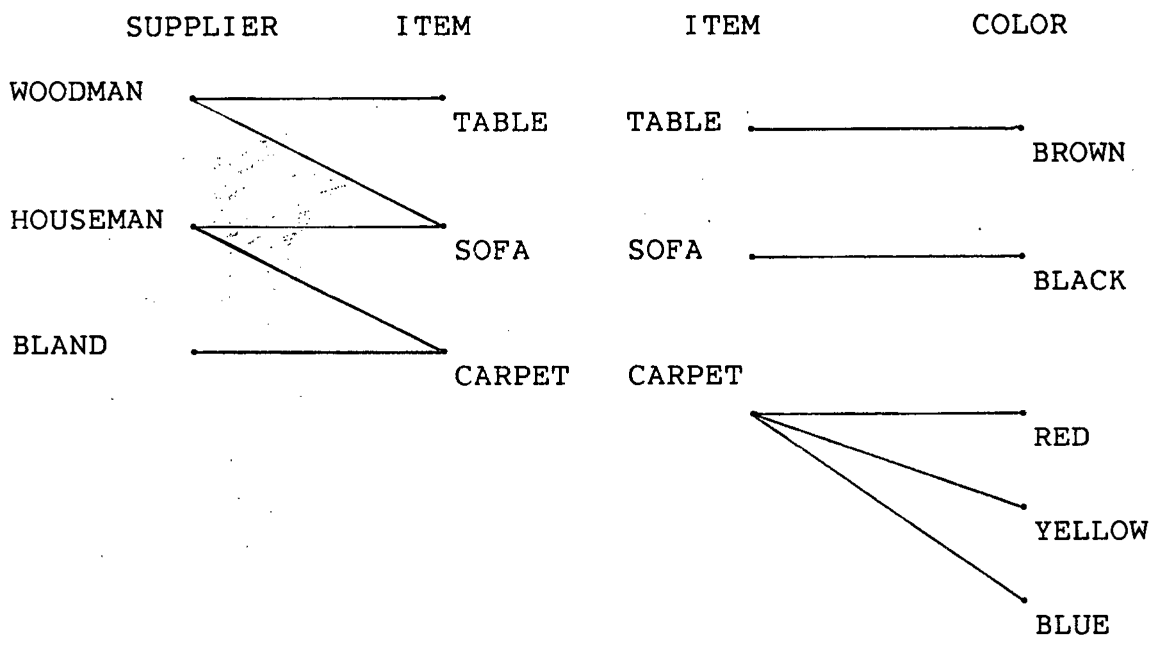

Figure 6. Mapping Diagram of the MVD in STOCK relation

be trivial if W = X u Y, that is when X and Y are dichotomies of R or when Y is a subset of X.

A Join Dependency (JD) is also a type of constraint that can be specified in a relation. A relation R(W) satisfies the JD*(X,Y,...,Z) if and only if it is the join of its projections on X,Y,...,Z, where X,Y,...,Z are subsets of attributes of R and W = X u Y u...u Z. From the definition of MVD, it can be observed that join dependency is a generalization of multivalued dependency. The relation in figure 7 illustrates join dependency. The original version appeared in (Date 1980).

S	P	J
s1	p1	j2
s1	p2	j1
s2	p1	j1
s1	p1	j1

Figure 7. Sample Relation for SPJ(S,P,J)

The JD*(SP,PJ,JS) holds in relation SPJ. If the pairs (S1,P1), (P1,J1) and (J1,S1) appear in SP, PJ and JS columns respectively, then the tuple (S1,P1,J1) must appear in SPJ. Thus at an instance when the relation contains the first two tuples, if a tuple (S2,P1,J1) is inserted then (S1,P1,J1) must also be inserted for the JD constraint to be satisfied. The JD is illustrated by the mapping diagram in figure 8.

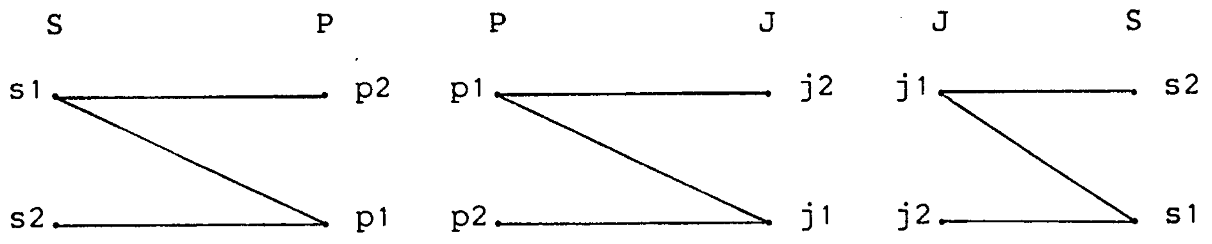

Figure 8. Mapping Diagram for the JD in SPJ relation

The relational database <u>normal</u> <u>forms</u> (Codd 1972, Fagin 1977a and others) are aimed mainly at eliminating certain anomalies in relations. The outputs from the design methodologies, presented in the next section, are evaluated in terms of the normal forms to which they conform. The normal forms are defined in section 4.1.

2.2 Design Approaches

A lot of work has been done on design methodologies for relational databases. Two of the earlier competing approaches are the third normal form decomposition of (Codd 1971) and the synthetic approach of Bernstein and others (Bernstein et al 1975 and Bernstein 1976). There is also the fourth normal form decomposition of (Fagin 1977a). Recently, (Zaniolo and Melkanoff 1981) proposed a decomposition approach not based on eliminating anomalies, but on complete data relatability. Other works on decomposition include those of (Delobel and Casey 1973) and (Delobel and Léonard 1974). Another approach is based on the Entity-Relationship model. Rules are presented in (Wong and Katz 1980) to transform a version of the Entity-Relationship model into relational database schemas.

A summary of the design approaches is given in figure 9. In the following paragraphs, a review of the literature on each of the methodologies is presented.

2.2.1 The Synthetic Approach

A major work on the synthetic approach to relational database design is that of (Bernstein 1976). Third normal form (3NF) relation schemas are synthesized from a given set of attributes and the functional dependencies among them. Since an initial relation is not assumed, the functional dependencies are

Approach[4]	Inputs	Outputs	Proponents[5]
Synthesis	FDs, Attributes	3NF relation schemas.	Bernstein, Swenson, Tsichritzis
Decomposition	FDs MVDs.	3NF 4NF.	Casey, Codd, Delobel, Fagin.
Decomposition/ Complete Data Relatability	Elementary FDs, multiple elementary MVDs.	3NF.	Melkanoff Zaniolo
Entity-Relationship/ Transformation Rules	Entities, Relationships and their Properties	4NF 4NF.	Katz, Wong.

Figure 9. Summary of Existing Design Approaches

defined as time-varying functions from one domain to another as stated in section 2.1.

The synthesis algorithm produces a nonredundant covering of the functional dependencies as follows. Let G be the set of FDs. The closure of G, denoted G^+, is the smallest superset of G that is closed under the following rules.

1.) Reflexivity (X--->X).

2.) Augmentation (if X--->Z then X u Y--->Z)

3.) Pseudotransitivity (if X--->Y and Y u Z--->W then X u Z--->W).

An FD $g \in G$ is said to be redundant in G if $G^+ = (G-\{g\})^+$. H is

[4]Other approaches are not radically different and should fit into one or more of the categories.

[5]The lists of the proponents are not exhaustive.

a nonredundant covering of a given set of FDs G if $G^+ = H^+$ and H contains no redundant FDs. The algorithm proceeds by removing extraneous attributes from the left sides of FDs in the nonredundant covering. An attribute Xi is extraneous in an FD g ϵ G, g:X1,...,Xp--->Y, if X1,...,Xi-1,Xi+1,...,Xp--->Y. Removing extraneous attributes helps to avoid partial dependencies and superkeys that are not keys.

If two relations have keys that are functionally dependent on each other, that is if they are equivalent, then the two relations can be merged together. The synthesis procedure accomplishes this fact by merging together groups of FDs if their left sides are functionally equivalent. The nonfunctional dependency between any two sets of attributes X and Y is represented by XY--->Ø, where Ø is a dummy attribute. The algorithm essentially creates one relation per nonfunctional relationship. Third normal form relation schemas are produced.

The uniqueness of FDs between any two sets of attributes have to be assumed because the treatment of the FDs is strictly syntactic. If there are two FDs on the same sets of attributes, then they are the same FD. Some problems resulting from the uniqueness assumption are illustrated by the following example. Let f1:DEPT#--->MGR# and f2:MGR#,FLOOR--->NO-OF-EMP be FDs such that f1 determines the manager of each department and f2 determines the number of employees working for a particular manager on a floor. By applying the pseudotransitivity rule to f1 and f2, we obtain f3:DEPT#,FLOOR--->NO-OF-EMP, which determines the number of employees of the manager on a particular floor. If we have another FD g:DEPT#,FLOOR--->NO-OF-

EMP which determines the number of employees of a department on a particular floor, then the FDs g and f3 are not the same if a manager can manage two departments. The attribute NO-OF-EMP of g may have to be renamed to make it distinct from the composition of f1 and f2. The problems associated with the uniqueness assumption is due mainly to the lack of expressiveness of FDs. It is not possible to tell whether a manager can manage more than one department from, f1:DEPT#--->MGR#. These problems are further discussed in section 2.3.

2.2.2 Decomposition

Codd's third normal form decomposition takes as input an initial set of relations along with a set of FDs (Codd 1971). Using the dependency information, the initial set of relation schemas is converted into 3NF schemas. For a relation to be in 3NF, there must not be a transitive dependency or partial dependency on a key. Therefore decomposition is carried out to eliminate transitive and/or partial dependency. In a relation schema $R(\underline{A},\underline{B},C,D)$, if A--->C holds then R is decomposed into $R_1(\underline{A},C)$ and $R_2(\underline{A},\underline{B},D)$ to eliminate the partial dependency of C on the key AB. In cases where there exist transitive dependencies (say A--->B and B--->C) in R(A,B,C,D), R is decomposed into $R_1(\underline{B},C)$ and $R_2(\underline{A},\underline{B},D)$. When both types of dependencies occur in R, a choice has to be made as to which decomposition should take place.

In (Fagin 1977a), multivalued dependency and fourth normal form decomposition were proposed. The 4NF decomposition is a generalization of the 3NF decomposition. The MVD models nonfunctional relationships between attributes. The design process takes a set of attributes along with a set of FDs and MVDs as input. A single relation schema consisting of all the attributes is formed. The basic rule is that if a FD X--->Y or a MVD X-->-->Y holds for a relation R(X,Y,Z), where Z is the set of attributes not in X or Y, then the relation can be decomposed into R_1(X,Y) and R_2(X,Z) without loss of information. In general, Xi-->-->Y_1|...|Yk holds for R(X,Y_1,...,Yk) if and only if R is the join of its projections R_1(X,Y_1),R_2(X,Y_2 provides a necessary and sufficient condition for a relation to be decomposable into some projections without loss of information. The decomposition process proceeds until no relation schema has nontrivial MVDs that are not functional dependencies. That is, every nontrivial MVD is implied by a key.

The 4NF decomposition approach provides a discipline for handling problems related to Bernstein's uniqueness assumption for functional dependencies. New attributes can be introduced in the initial relation schema and renamed after a decomposition. The uniqueness assumption need only hold within each relation in the net design. However, the 4NF decomposition is also faced with a number of problems. The MVDs are part of the input to the design process, but they are not easily recognizable within a relation. The order in which decomposition is carried out also affects the design. But choice of decomposition is only based on heuristics. It is not clear how to relate order of decomposition to optimal design or what constitutes an optimal design.

(Rissanen 1977) proposed the notion of independent components of relations in deciding when a representation is "good".

Projections of a relation R provide a good representation of R if all the dependencies in R are "nicely" embedded in the projections. (Zaniolo and Melkanoff 1981) also deals with these problems by decomposing for <u>complete</u> <u>data</u> <u>relatability.</u>

2.2.3 Decomposition/Complete Data Relatability

In (Zaniolo and Melkanoff 1981), a new approach to the design of relational databases based on Complete Relatability of Data was proposed. Since different decomposition paths can produce different quality of relations even when they all conform to the same normal form, eliminating anomalies seems not to be an adequate criterion for database design. The following example illustrates the point. A relation schema R(<u>AC#</u>,EM,TX) relates employees and their telephone extensions to the accounts which they manage. Assuming an account has only one manager and an employee has only one telephone extention, then the functional dependencies AC#--->EM and EM--->TX hold in R. The FD AC#--->TX can be inferred by the transitivity rule. Hence the relation can be decomposed into

1.) R_1(<u>AC#</u>,EM) and R_2(<u>EM</u>,TX) based on the FD EM--->TX

2.) R_1(<u>AC#</u>,EM) and R_2(<u>AC#</u>,TX) based on the FD AC#--->EM.

The resulting schemas in both cases are in Boyce-Codd normal form, but (1) is a better representation because the transitive FD AC#--->TX can be inferred from (1) but it is concealed in (2). The schemas in (1) are said to ensure compl

The design approach assumes an initial set of database relation schemas with sample relations from which the dependencies are detected. <u>Elementary</u> <u>FDs</u> and <u>multiple</u>

elementary MVDs are introduced to simplify the task of detecting the dependencies in relations. They constitute a small subset of all FDs and MVDs in a relation and they have nondecomposable structures. All other FDS and MVDs can be inferred from them. Elementary FDs and MVDs are generated as follows: For a given relation R(W), a partial order is defined among ordered pairs of subsets of W such that

$(X_1,Y_1) \leq (X_2,Y_2)$ if $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$.

Let G be the set of MVDs. The minimum members of G are the elementary MVDs of R, and are denoted G*. Thus X-->-->Y is elementary if and only if $Y \cap X = \emptyset$ and there exists no distinct MVD X'-->-->Y' where $X' \subseteq X$ and $Y' \subseteq Y$. F*, the elementary FDs for a set F of FDs, is obtained similarly. The elementary MVDs are further subdivided into single and multiple elementary MVDs. Single elementary MVDs are those that do not have the same left side with any other elementary MVD, while multiple elementary MVDs have one or more other MVDs with the same left side. The concept of elementary MVDs helps the designer in characterizing the dependency structure in a given relation. There is an algorithm for generating all multiple elementary MVDs with left side X if an elementary MVD with a left side Y is known such that $Y \subset X$. There is also an algorithm for generating the multiple elementary MVDs which form the minimum cover for the MVDs with respect to the reflexivity, augmentation, additivity and complementation rules for functional and multivalued dependencies. The reference rules for functional and multivalued dependencies are discussed in (Beeri et al 1977).

The decomposition algorithm recursively decomposes relations into a pair of subprojections according to multiple elementary MVDs, until a set of atomic relations is obtained. An atomic relation contains only trivial MVDs. At completion, the algorithm produces a set of atomic subrelations and a set of FDs referred to as ACOVER and ZCOVER respectively. The two sets are later used in constructing 3NF relations. The initial relation is reconstructable as the natural join of the projections, thus the decomposition is content preserving. Complete data relatability also demands that the structure (i.e the attribute set, the FDs and the MVDs) of the original relations be preserved. The algorithm selects those elementary MVDs that ensure preservation of the structural information. In testing the data relatability condition, the notions of <u>admissibility of FD covers</u> and <u>scope</u> of elementary FD are introduced. The scope of an FD X--->A is the set X u {A}. A ZCOVER generated from a relation R is said to be admissible with respect to ACOVER

1.) If the ZCOVER contains an elementary FD with scope X, it must contain every other elementary FD of R with scope X. Moreover, if R[X] for such X is atomic, then the ACOVER must contain it as a member.

2.) If ACOVER contains an atomic projection R[X], then the ZCOVER must contain every elementary FD of R having scope X.

In a decomposition of R(Z) into $R_1(Y)$ and $R_2(X)$, let F1 and F2 denote the elementary FDs in R_1 and R_2 respectively. The FDs with scope Z in R can not be inferred by F1 and F2; they are explicitly entered into ZCOVER. The remaining FDs in R(Z), F*, are preserved by selecting a decomposition such that F* can be inferred from the elementary FDs in the projections; that is $F^* \subseteq (F1 \cup F2)^+$. This is the complete relatability condition (CRC) for elementary FDs. To avoid redundancy, decomposition is based on the multiple elementary MVDs. Hence, the CRC for MVDs must also be satisfied. However, the reverse projectability rule does not hold for MVDs. That is, X-->-->Y in a projection of R does not imply X-->-->Y in R. A weaker property known as joinability is used to generate the set of MVDs inferable from the elementary MVDs in the projections.

The two resulting sets, ACOVER and ZCOVER, can be used to improve 3NF relations using Bernstein's algorithm. The ZCOVER constitutes a minimal cover for FDs in the original relation R and can be used as minimum FD cover in Bernstein's algorithm. The algorithm will now produce 3NF relations which completely characterize the functional relationship in the initial relation. The nonfunctional relationships are represented by elements of ACOVER without corresponding FDs in the ZCOVER. Each of them form a separate relation.

The Decomposition/Complete data relatability have been able to deal with some of the problems in normal decomposition. The complete relatability condition is able to guide the order in which decomposition is carried out to produce subrelations that preserve the dependencies in the original relation. Characterizing the MVDs have been made easier, but detecting the initial MVDs from which others can be generated is not trivial. Like most decomposition methods, a set of initial relations is assumed. The universal relation assumption is known to have some undesirable consequences. In section 2.3, some of these consequences, as in (Kent 1981), are discussed.

2.2.4 Entity-Relationship Model/Transformation Rules

(Wong and Katz 1980) proposed a variant of the Entity-Relationship model as an intermediate design model which is in turn transformed into relation schemas. The following semantic objects are recognized within the intermediate model.

1.) Entity sets (E(t)): a one parameter family of sets which changes as members are inserted and deleted.

2.) A Property of an entity set E(t) is a function f_t mapping E(t) to some set V of values at time instance t. The function is defined on all of E(t), and for every e ϵ E(t), $f_t(e)$ is unique.

3.) Relationships: A relationship R_t among entity sets E1(t),...,En(t) is a subset of the cartesian product $E_1(t) \times E_2(t) \times ... \times En(t)$. No relationship is derivable from other relationships; they are independent and nondecomposable.

4.) Properties of relationships: In a similar fashion to entity sets, properties can be defined on relationships.

5.) Single-valued binary relationships: A binary relationship R_t on entity sets $E_1(t)$ and $E_2(t)$ is single-valued if each entity in $E_1(t)$ occurs in at most one pair in R_t.

6.) Associations: An association is a binary relationship in which $E_1(t)$ entities occurs in exactly one instance. No properties are allowed on associations or single-valued relationships.

The design goal is to prevent "update anomalies". An update anomaly is defined as either a fragmentation of the atomic operations or uncontrolled side effects. An atomic operation is one of the following:

1.) Inserting or deleting an entity.

2.) Inserting or deleting an instance of a relationship.

3.) Changing the value of a function (property or association of an entity).

Deletion or insertion of an entity do have side effects. The deletion of an entity causes a deletion of any instance of a relationship in which it participates. Any entity that has the deleted entity as its range value in an association is also deleted. The insertion of an entity "e" requires the entity that is the value of any association of e to already exist. The order of insertion of an entity may be constrained by associations. Thus, a cycle of associations is not allowed.

The intermediate model is transformed into relation schemas by the following rules.

1.) Each entity set has an explicit identifier which represents it globally in the model. An identifier is a one-to-one property of an entity set.

2.) The identifier of a primitive object together with all its primary functions are grouped in the same relation. A primary function is a property or an association, and a primitive object is either a relationship or an entity set in its role as the domain of a primary function.

3.) There is one and only one primitive object per relation.

The transformation rules are such that they preserve the atomicity of updates and control the side effects. Rule 2 groups entities together with their corresponding properties and associations in the same relation. It allows deletion and insertion of an entity to be made along with its associations and properties in a single relation tuple. A violation of one of the normal forms (1NF,2NF,3NF,4NF) can be interpreted as a violation of one of the rules (Wong et al 1980).

1.) The grouping together of two primitive objects with no entity in common or a function of an entity and a relationship involving it, both result in a relation that is not in 2NF.

2.) Putting two functions f1 and f2 with different entity sets as their domains in the same relation can violate the 2NF. If the functions are of the form E_1--->E_2--->S, 3NF is violated.

3.) The grouping of two relationships with a common entity set together in the same relation can produce a result that is not in 4NF.

The schemas resulting from the rules, therefore, conform to the 4NF. The following example illustrates the design process.

Entity sets	Properties
EMP	ENAME, BIRTHYR
DEPT	DNAME, LOCATION
JOB	TITLE, SALARY

Associations

Works-in(EMP, DEPT)

Assignment(EMP, JOB)

Relationships	Status	Properties
mgr(DEPT, EMP)	single-valued	nil
qualified(EMP, JOB)	general	nil
allocation(DEPT, JOB)	general	number

The intermediate model consists of the following primitive objects and functions.

Primitive Objects	Functions
EMP	ENAME, BIRTHYR, works-in, assignment
DEPT	DNAME, LOCATION
JOB	TITLE, SALARY
mgr	-
allocation	NUMBER
qualified	-

The model is transformed into five relations as follows.

EMP(ENO,ENAME,BIRTHYR,ASSIGNMENT,EDEPT)

DEPT(DNO,LOCATION,MGR)

JOB(JID,TITLE,SALARY)

ALLOC(DNO,JID,NUMBER)

QUAL(JID,ENO)

The Entity-Relationship/Transformaton rules provide a practical approach to relational database design. But as noted by the authors, every 4NF relation schema is not necessarily generated from the intermediate model via the mapping rules. The resulting schema is restricted by the intermediate model. We believe that the explicit specification of whether the relationships and funtions are partial or total on the source and target sets, provides more meaning to the intermediate model. The additional meaning allows more relaxed rules to be defined. These points will become more evident in our approach, presented in chapter 3.

2.3 Appraisal of the Design Approaches

The decomposition approaches to relational database design share a lot in common. In particular, they take as input an initial relation design. Recently, the consequences of the Universal relation and other assumptions were discussed in (Kent 1981). In addition to the specific questions raised in each of the approaches, the paper provides a common ground for appraising both the synthetic and decomposition methods. A number of questions concerning the Entity-Relationship/Transformation rules are raised in section 2.2.4. The design method is radically different from the others.

2.3.1 Consequences of Universal Relation Assumption

Both the decomposition and synthetic methods make certain implicit assumptions:

1.) There are no domains: Columns of relations are distinctly named with no facility for stating the underlying domains that might be common to several columns.

2.) A join compares columns if and only if they have the same name.

An explicit assumption common to most decomposition methods is the Universal relation assumption. For a given set of relations, S = $\{R_1(X_1),\ldots,Rn(Xn)\}$, a universal relation U(T) exists such that

1.) The column names of U consists of all the column names of the relations in R, that is $T = X_1 \cup X_2 \cup \ldots \cup Xn$.

2.) Each relation in S is a projection of U.

However, these assumptions have implications that are not compatible with practical database design.

The universal relation assumption implies that columns have the same meaning in every relation, because they are projected from the same source. Therefore, wherever an attribute occurs, it must necessarily have the same extensions. Hence, updates to relations of the form $R_1(X,Y)$ and $R_2(X,Z)$ must preserve equality of the projections $R_1[X]$ and $R_2[X]$. In essence, we can not use the same attribute with different intensions in various relationships. It is also not meaningful to have two relations with identical column names.

In Berntein's synthesis, attributes are allowed to be renamed in order that the uniqueness assumption for FDs be preserved. Decomposition methods do not explicitly require the uniqueness of FDs, but attributes can also be renamed after the decomposition of a relation. One consequence of renaming attributes is that it is not possible to make natural joins over such distinct attributes even though they share the same underlying domain. Though in practice some systems allow joins over different column names, this is only useful if there is a provision for checking that the column names have the same underlying domain. A closely related problem is that of expressing relationships existing among attributes of a relation.

2.3.2 Data Dependencies

Expressing the dependencies between data objects is very crucial to database design. The synthetic approach takes as input a set of FDs, but nonfunctional relationships can not be represented directly. The 4NF decomposition allows nonfunctional relationships to be expressed as MVDs. However, a MVD is defined such that it is only recognizable when it coexist with another one in the same relation. The task of detecting MVDs within a relation is also not trivial. The MVDs are not only unintuitive, their properties are not well understood. Some MVDs hold in a projection of a relation but not in the original relation. These are referred to as embedded MVDs. An issue yet unresolved is whether there exist inference rules (from projections to the join) stronger than the joinability (Zaniolo and Melkanoff 1981). Thus, multivalued dependency is not very suitable as a means of representing many-many relationships. In general, two many-many relationships E----S and E----D, will not appear as multivalued dependencies if there also exists some relationships involving S and D (Kent 1981). Actually, in a relation schema R(ESD) containing attributes with the two many-many relationships, if S and D have some relationsips, we have a join dependency constraint. Clearly, the dependencies are not enough to specify the relationships that do exist among data items of a database.

2.3.3 Decomposition versus the Synthetic Approach

The decomposition and synthetic methods differ mainly in the type of input they take. In general, decomposition takes as input an initial set of relations, FDs and MVDs. Fagin's 4NF decomposition accepts sets of attributes, FDs and MVDs, but the first step converts all the attributes into a single relation. Only functional dependencies can be specified directly in synthetic methods, because MVDs can only be defined within the context of a relation. The MVDs would have to be specifiable in a context-independent form, if the synthetic methods are to accept them directly as input.

Since synthetic methods do not take initial relations as input, column names are necessarily unique. This is also the case for decomposition when the universal relation is assumed. With the present state of dependency theory, none of the approaches is clearly superior to the other. Synthesis appears more desirable in practice, especially for large databases. Decomposition tends to leave residue relations which sometimes model relationships that can not be expressed as FDs or MVDs. But sometimes the attributes of such relations do not bear any direct relationship.
Kent is of the opinion that a more extensive dependency theory, in which all dependencies could be formally expressed, is needed. With such a theory, the synthetic approach might be preferred. Relations capturing all the relationships would be generated, while decomposition would continue to leave unrelated elements aggregated in residue relations (Kent 1981).

CHAPTER III . The Full Mapping Approach

This chapter deals with Full Mappings and the transformation rules. The mappings are proposed as an alternative to functional and mutivalued dependencies. The transformation rules generate relation schemas from the full mapping specifications.

3.1 Full Mapping Specification

A formal information analysis of an application environment reveals relevant entity sets, value sets and associations (section 2.1.1). We place emphasis on the types of mapping that exist among the sets. By representing properties with appropriate attribute-names and the entity sets with a primary attribute, the associations can all be expressed as mappings between attributes. A primary attribute has to be a property that provides a one-to-one correspondence between the entity set and the property-value set. Hitherto, in relational database theory, data relationships are expressed as functional, multivalued or join dependencies. Full Mapping is proposed as a means of specifying relationships between data items.

3.1.1 Mapping Types

Let A and B be sets acting as source and target of a mapping respectively. The following mapping types can be defined.

1.) One-one mapping (A 1----1 B): There is a one-to-one correspondence between the source and the target elements.

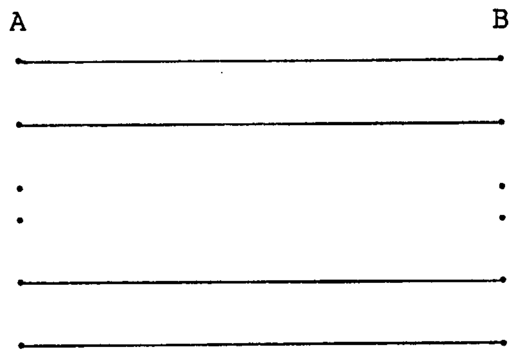

Figure 10. One-to-one Mapping Diagram

2.) Many-one mapping (A m----1 B): Disjoint sets of A-elements are mapped to unique elements in B.

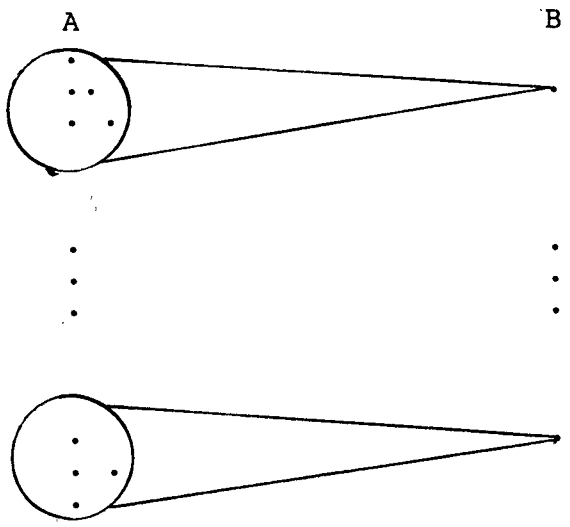

Figure 11. Many-one Mapping Diagram

A one-many mapping from A to B can always be treated as a many-one mapping from B to A. Hence we do not have to distinguish between many-one and one-many mapping types.

3.) Many-many mapping (A m----n B): Elements in A are mapped to sets in B, but the B-sets are not necessarily disjoint.

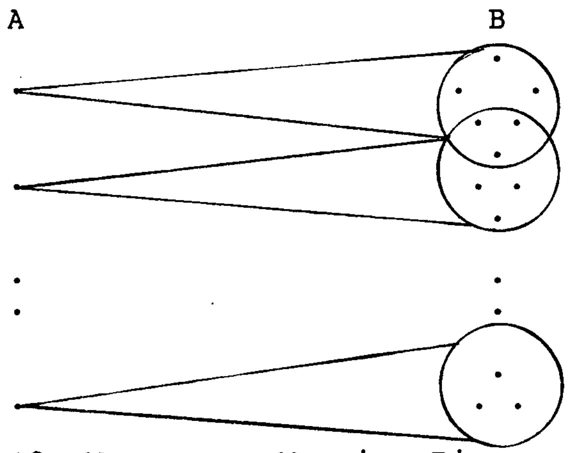

Figure 12. Many-many Mapping Diagram

A mapping, apart from being one-one, many-one or many-many, is either total or partial on the source and target sets. An <u>into-mapping</u> on a set expresses the fact that an element in the set may not be involved in the mapping. There is an element of relativity in deciding whether a mapping is 'into' or 'onto' a set. Let us consider an application environment where a supplier (SNO) of an item stays in a particular CITY and the cities have STATUS attached to them. Every supplier stays in a city and every city has a status. The mappings SNO----CITY and CITY----STATUS appear to be both total on their source and target sets. However, if we wish it possible to enter the information that a particular city has a particular status even when there are no suppliers located in that city, then the mapping SNO----CITY is 'into' CITY. That is, at a database instance, if we match the extension of CITY to that of SNO there may be cities not mapped to any supplier. An into-mapping on a set is a generalized form of the conditional association in (Raver and Hubbard 1977). The conditional association expresses the case where an element in the source has exactly one corresponding target element or none at all.

An onto-mapping on a set is a total mapping on the set. A 'onto' a set if every element of the set always participate in the mapping. By the definition of into and onto mappings, it is not possible to have two onto-mappings on the same set with different domain extensions at any database instance.

There is inherent semantic information expressed when the into/onto status of a mapping is stated. This should be combined with the mapping types to provide more meaning to data dependencies. Hence, there are a total of twelve mapping types which we refer to as full mappings.

An example of a fully defined mapping between two sets A and B is many-many (onto, into) written as A^{o} m----n B^{i}.

3.2 Design of Relation Schemas from Full Mappings

A mapping between two sets X and Y is a binary relation between the sets. It can be represented by a relation schema R(X,Y). Such a schema is atomic since the mappings are nondecomposable and are not derivable from other mappings. If the principal schemas are expressed exclusively as atomic relations, then there will be the need to apply n-ary joins to obtain higher degree relations in order to define views and to represent a broad class of queries. Therefore, the rules are defined to detect the mapping types that can be combined. The relational schema design problem is to avoid repeating attributes in a large number of low degree relations, as well as avoid the problems that could arise from joins.

3.2.1 Transformation Rules for BCNF schemas

A relation schema R is in Boyce-Codd normal form (BCNF) if for all disjoint and nonempty sets of attributes X and Y in R, if X--->Y then X is a superkey of R (Beeri and Bernstein 1979). Hence, the rules are such that every determinant is a candidate key. Every determinant is relevant in determining the BCNF schemas. Therefore, we shall include superkeys in the set of candidate keys. Since proper keys are superkeys, any claim made for the set of candidate keys of R is also true for the proper keys of R.

1.) Exclusive mappings are those that have unique attributes; they remain uncombined. The corresponding atomic relation schemas are in their final form.

Let X and Y be sets of attributes. A mapping from X to Y can be transformed into a relation schema R(X,Y), regardless of whether the mapping is 'into' or 'onto' X and Y. The candidate keys are determined as follows: The convention adopted is to leave the into/onto status of a mapping unspecified if the rule is valid for either case.

a) X 1----1 Y key(R) = {X,Y}

b) X m----1 Y key(R) = {X}

c) X m----n Y key(R) = {XY}

2.) Common Attribute Groups: The nonexclusive mappings are arranged into groups, such that every mapping in a group has a common attribute with at least one other mapping in the group. There are no two groups having an attribute in common. Within a common attribute group, two mappings from A to B and from B to C can be combined into a relation schema R(A,B,C) according to the following rules.

a) A 1----1 B^{o} + B^{o} 1----1 C key(R) = {A,B,C}

b) A 1----1 B^{o} + B^{o} m----1 C key(R) = {A,B}

c) A m----n B^{o} + B^{o} m----n C key(R) = {ABC}

d) A 1----m B^{o} + B^{o} m----1 C key(R) = {B}

3.) The resulting schemas from rule (2) can be combined successively with other mappings in the group as follows:
Let R_1(X,A) be a re schema with sets of attribute X and an attribute A. R_1 can be combined with a mapping from A to B into R_2(X,A,B). There are three relevant cases depending on whether B is contained in X. If B is contained in X and X = Y u B, the resulting schema is R_2(Y,A,B). If key(R_1) is the set of candidate keys of R_1, the candidate keys, key(R_2), of R_2 is determined according to the following rules:
Case (i) A $\in$ key(R_1) and B $\not\subset$ X

a) R_1(X,A^{o}) + A^{o} 1----1 B; key(R_2) = key(R_1) u {A,B}

b) R_1(X,A^{o}) + A^{o} m----1 B; key(R_2) = key(R_1) u {A}

Case (ii) AB $\in$ key(R_1) and B $\subseteq$ X. This is equivalent to joining over structured entities.

a) R_1(Y,A^{o},B^{o}) + A^{o} 1----1 B^{o}; key(R_2) = key(R_1) u {A,B}

b) R_1(Y,A^{o},B^{o}) + A^{o} m----1 B^{o}; key(R_2) = key(R_1) u {A}

c) R_1(Y,A^{o},B^{o}) + A^{o} m----n B^{o}; key(R_2) = key(R_1)

Case (iii) AB $\not\subseteq$ key(R_1) and B $\subseteq$ X.

a) $R_1(Y,\overset{o}{A},\overset{o}{B}) + \overset{o}{A}$ m----n $\overset{o}{B}$; key(R_2) = key(R_1)

4.) Exception to the rules.

All the rules require that the common attribute(s) in a join be mapped 'onto' in their corresponding mappings. However, the rules are valid if all the necessary 'ontos' are replaced by 'intos', as long as we can guarantee semantically that the extensions of the attributes involved will be equal at any database instance. Those attributes are said to have equivalent domain extensions. In essence, onto-mappings guarantee equivalent domain extensions of join attributes.

3.2.2 Transformation Rules for PJ/NF Schemas

1.) The exclusive mappings are transformed into relation schemas as in the BCNF rules.

2.) Common Attribute Groups: Within the common attribute groups, the following rules hold.

a) A 1----1 $\overset{o}{B}$ + $\overset{o}{B}$ 1----1 C key(R) = {A,B,C}

b) A 1----1 $\overset{o}{B}$ + $\overset{o}{B}$ m----1 C key(R) = {A,B}

c) A 1----m $\overset{o}{B}$ + $\overset{o}{B}$ m----1 C key(R) = {B}

3.) The resulting schemas from rule (2) can be combined successively with other mappings in the group as follows:

Case (i) $A \in key(R_1)$ and $B \not\subseteq X$

a) $R_1(X,\overset{o}{A}) + \overset{o}{A}$ 1----1 B; $key(R_2) = key(R_1) \cup \{A,B\}$

b) $R_1(X,\overset{o}{A}) + \overset{o}{A}$ m----1 B; $key(R_2) = key(R_1) \cup \{A\}$

Case (ii) $AB \in key(R_1)$ and $B \subseteq X$. Let $Y = X - B$.

a) $R_1(Y,\overset{o}{A},\overset{o}{B}) + \overset{o}{A}$ 1----1 $\overset{o}{B}$; $key(R_2) = key(R_1) \cup \{A,B\}$

b) $R_1(Y,\overset{o}{A},\overset{o}{B}) + \overset{o}{A}$ m----1 $\overset{o}{B}$; $key(R_2) = key(R_1) \cup \{A\}$

c) $R_1(Y,\overset{o}{A},\overset{o}{B}) + \overset{o}{A}$ m----n $\overset{o}{B}$; $key(R_2) = key(R_1)$

4.) The exception to the rules in section 3.3.1 also holds for projection-join normal form schemas.

3.3 Basis for the Transformation Rules

Combining two or more mappings into a relation schema is equivalent to joining the corresponding atomic relations of the mappings. Thus, the rules must at least ensure that the joins are lossless.

3.3.1 Necessary and Sufficient Condition for a Lossless Join

A join is lossless (in the synthesis context) if the resulting relation can be projected back to the original relations before the join. That is, if relation schemas R(X,Y) and R(Y,Z) are joined over Y into R(X,Y,Z), the join is lossless if the projections of R(X,Y,Z), R[X,Y] and R[Y,Z], are equal to R(X,Y) and R(Y,Z) respectively.

It has been observed in (Codd 1979) that joins lose information when the relations involved do not have equal projections on their common attribute(s). The observation is not just an extensional concept; it reveals an important semantic notion.

<u>Claim:</u> A join of relation schemas is lossless if and only if the common attributes are mapped 'onto' in their corresponding mappings or the attributes always have identical domain extensions.

<u>Proof</u> (Sufficiency):

Let $R_1(X,Y)$ and $R_2(Y,Z)$ be relation schemas denoting two mappings M1(X----Y) and M2(Y----Z) respectively. The join of $R_1(X,Y)$ and $R_2(Y,Z)$ over Y is $R(X,Y,Z) = \{(x,y,z): (x,y) \in R_1(X,Y) \text{ and } (y,z) \in R_2(Y,Z)\}$. If the extensions of Y in R_1 and R_2 are always equal, then at any database instance, for each "y" in the (x,y)-tuples of $R_1(X,Y)$ there exists at least one identical "y" in the (y,z)-tuples of $R_2(Y,Z)$. If a particular "y" occurs n and m times in the Y-columns of $R_1(X,Y)$ and $R_2(Y,Z)$ respectively, then the "y" will occur n x m times in the Y-column of R(X,Y,Z). Hence, every (x,y) and (y,z) pairs of $R_1(X,Y)$ and $R_2(Y,Z)$ will appear in the XY-column and YZ-column of R(X,Y,Z), respectively, at least once. Therefore, a projection of R(X,Y,Z) over XY and YZ will reproduce the original relations $R_1(X,Y)$ and $R_2(Y,Z)$. Repeated tuples are merged since projection is a set operation.

<u>Proof</u> (Necessary Condition):

Let R(X,Y,Z) be the join of $R_1(X,Y)$ and $R_2(Y,Z)$ as in the sufficiency proof above. If at any database instance the domain extension of Y in R_1 is not equal to that of R_2, there will either be a y_i in the Y-column of R_1 not in the Y-column of R_2 or a y_j in the Y-column of R_2 not in that of R_1. Hence, a tuple (x_i, y_i) or (y_j, z_j) will not appear in the XY-column or YZ-column of R(X,Y,Z) respectively. Therefore, R[X,Y] and R[Y,Z], the projections of R(X,Y,Z) over XY and YZ will not be equal to $R_1(X,Y)$ and $R_2(Y,Z)$. Thus, the join will not satisfy the lossless property.

3.3.2 The Candidate Keys

The candidate keys in the transformation rules in section 3.3 are derived according to the following claim:

Claim: Let the set of candidate keys of the relation schemas $R_1(X,Y)$ and $R_2(X,Z)$ be key(R_1) and key(R_2) respectively. The new set of candidate keys key(R_3), after a lossless join of R_1 and R_2 over the set of attribute X, depends on whether X is a candidate key in R_1 or R_2. There are four cases.

1.) If $X \in key(R_1)$ and $X \notin key(R_2)$ then $key(R_3) = key(R_2)$.

2.) If $X \notin key(R_1)$ and $X \in key(R_2)$ then $key(R_3) = key(R_1)$.

3.) If $X \in key(R_1)$ and $X \in key(R_2)$ then $key(R_3) = \{key(R_1) \cup key(R_2)\}$

4.) If $X \notin key(R_1)$ and $X \notin key(R_2)$ then $key(R_3) = \{key(R_1) \times key(R_2)\}$

proof

Case (1) $X \in key(R_1)$ and $X \notin key(R_2)$.

In a lossless join of $R_1(X,Y)$ and $R_2(X,Z)$ into $R_3(X,Y,Z)$, each tuple of R_2 will appear once in the XZ-column of R_3. This is the case because for every x-value in R_2 (not necessarily unique since $X \notin key(R_2)$), there is a unique x-value in R_1. Hence, the number of entries in R_3 is determined by tuples of R_2. Therefore, tuples of R_3 are determined uniquely by the keys of R_2.

Case (2) Similarly, when $X \notin key(R_1)$ and $X \in key(R_2)$, the keys of R_1 determine tuples of R_3 uniquely.

Case (3) $X \in key(R_1)$ and $X \in key(R_2)$.

There is a one-to-one correspondence between tuples of R_1 and R_2. Hence tuples of R_3 are direct concatenation of tuples of R_1 and R_2 over equal x-values. Therefore, the tuples of R_3 are uniquely determined by the keys in $key(R_1)$ or $key(R_2)$.

Case (4) $X \notin key(R_1)$ and $X \notin key(R_2)$.

In general, every x-value in R_1 or R_2 can appear more than once. For any x-value with n and m entries in R_1 and R_2, respectively, tuples containing the particular x-value will appear n x m times in R_3. Since X is not a key in R_1 or R_2, there is at least an x-value in each of the X-columns of R_1 and R_2 that appears more than once. The cardinality of R_3 is always greater than either of R_1 or R_2. Therefore, none of the keys in $Key(R_1)$ or $key(R_2)$ can uniquely determine tuples of R_3. It is only a combination of a key in $key(R_1)$ and one in $key(R_2)$ that determines tuples of R_3 uniquely. The new set of candidate keys is the cartesian product of $key(R_1)$ and $key(R_2)$.

3.3.3 The BCNF Rules: Verification

The schemas resulting from the transformation rules can be grouped into two categories; the atomic relation schemas and those from rules (2) and (3). The candidate keys of an atomic schema R(X,Y) for X 1----1 Y, X m----1 Y and X m----n Y are {X,Y}, {X} and {XY} respectively. In each case, the determinants are also candidate keys. The atomic schemas are trivially in BCNF.

A combination of two or more mappings in rules (2) and (3) is equivalent to a join of their corresponding atomic relations. The rules are formulated, such that the join is lossless and every determinant is a superkey. We ensure lossless join by combining over attributes which have equivalent domain extensions in their corresponding mappings. The join attributes are either mapped 'onto' or they are involved in 'into' mappings that always have the same extensions.

In Rule (2), there are eleven distinct combinations. Only four of them satisfy the BCNF condition.

Let A----B and B----C be two mappings to be combined into a relation schema R(A,B,C). The set of determinants of R, Det(R), and key(R) for the various combinations are as follows.

	Det(R)	Key(R)
1.)* A 1----1 $\overset{o}{B}$ + $\overset{o}{B}$ 1----1 C	{A,B,C}	{A,B,C}
2.)* A 1----1 $\overset{o}{B}$ + $\overset{o}{B}$ m----1 C	{A,B}	{A,B}
3.) A 1----1 $\overset{o}{B}$ + $\overset{o}{B}$ m----n C	{A,B}	{BC}
4.)* A 1----m $\overset{o}{B}$ + $\overset{o}{B}$ m----1 C	{B}	{B}

5.) A m----1 B^{o} + B^{o} 1----1 C	{A,B,C}	{A}
6.) A m----1 B^{o} + B^{o} m----1 C	{A,B}	{A}
7.) A m----1 B^{o} + B^{o} 1----m C	{A,C}	{AC}
8.) A m----1 B^{o} + B^{o} m----n C	{A}	{ABC}
9.) A m----n B^{o} + B^{o} 1----1 C	{B,C}	{AB}
10.) A m----n B^{o} + B^{o} m----1 C	{B}	{AB}
11.)* A m----n B^{o} + B^{o} m----n C	{Ø}	{ABC}

The combinations where Det(R) $\subseteq$ Key(R) produce schemas that conform to the Boyce-Codd normal form. The combinations with asterisks satisfy this condition. They are the only ones allowed in rule (2).

Rule (3) allows successive combination of mappings with schemas generated by rule (2). If $R_1(X,A)$ is to be combined with a mapping A----B into $R_2(X,A,B)$, there are four cases to be considered.

Case (1) $B \not\subseteq X$ and $A \in key(R_1)$

Case (2) $B \not\subseteq X$ and $A \notin key(R_1)$

Case (3) $B \subseteq X$ and $AB \in key(R_1)$

Case (4) $B \subseteq X$ and $AB \notin key(R_1)$

Let $Det(R_1)$ and $Key(R_1)$ be the set of determinants and the set of candidate keys of R_1 respectively. The set of determinants and the set of candidate keys of R_2 for the various combinations are as follows:

For cases (3) and (4), let Y = X - B.

Case (1)	$Det(R_2)$	$Key(R_2)$
a)* $R_1(X,A^o) + A^o$ 1----1 B	$Det(R_1)$ u {B}	$Key(R_1)$ u {B}
b)* $R_1(X,A^o) + A^o$ m----1 B	$Det(R_1)$	$Key(R_1)$
c) $R_1(X,A^o) + A^o$ m----n B	$Det(R_1)$	{AB}
d) $R_1(X,A^o) + A^o$ 1----m B	$Det(R_1)$ u {B}	{B}

Case (2)	$Det(R_2)$	$Key(R_2)$
a) $R_1(X,A^o) + A^o$ 1----1 B	$Det(R_1)$ u {A,B}	$Key(R_1)$
b) $R_1(X,A^o) + A^o$ m----1 B	$Det(R_1)$ u {A}	$Key(R_1)$
c) $R_1(X,A^o) + A^o$ m----n B	$Det(R_1)$	$Key(R_1)$ x {AB}
d) $R_1(X,A^o) + A^o$ 1----m B	$Det(R_1)$ u {B}	$Key(R_1)$ x {B}

Case (3)	$Det(R_2)$	$Key(R_2)$
a)* $R_1(Y,A^o,B^o) + A^o$ 1----1 B^o	$Det(R_1)$ u {A,B}	$Key(R_1)$ u {A,B}
b)* $R_1(Y,A^o,B^o) + A^o$ m----1 B^o	$Det(R_1)$ u {A}	$Key(R_1)$ u {A}
c)* $R_1(Y,A^o,B^o) + A^o$ m----n B^o	$Det(R_1)$	$Key(R_1)$

Case (4)	Det(R_2)	Key(R_2)
a) $R_1(Y,A^{o},B^{o}) + A^{o}$ 1----1 B^{o}	Det(R_1) u {A,B}	Key(R_1)
b) $R_1(Y,A^{o},B^{o}) + A^{o}$ m----1 B^{o}	Det(R_1) u {A}	Key(R_1)
c)* $R_1(Y,A^{o},B^{o}) + A^{o}$ m----n B^{o}	Det(R_1)	Key(R_1)

Given that Det(R_1) $\subseteq$ Key(R_1), the combinations with asterisks are such that Det(R_2) $\subseteq$ Key(R_2). The combinations form rule (3). Joins are both commutative and associative (Aho et al 1979), therefore the order of combination within the groups is immaterial.

3.3.4 The PJ/NF Rules: Verification

A join dependency constraint JD*(X,Y,...,Z) in a relation schema R, where X,Y,...,Z are combinations of attributes of R, states that R is the join of its projections over X,Y,...,Z. A relation R is in projection-join normal form (PJ/NF) if and only if every join dependency is implied by a candidate key of R. A JD*(X,Y,...,Z) in R is implied by candidate keys of R if the join attributes in X,Y,...,Z uniquely determine tuples of R. The atomic relations are trivially in PJ/NF since the full mappings are nondecomposable and no mapping can be derived from some other mappings.

The rules in (2) and (3) ensure that the joins are lossless and that the join attributes of all the JDs in a resulting schema are candidate keys in the schema.

Let $R(X_1 \cup \ldots \cup X_n)$ be a relation schema resulting from a combination of n mappings represented by the atomic schemas $R_1(X_1), \ldots, R_n(X_n)$ respectively. Let $\{Y_1, \ldots, Y_{n-1}\}$ be the join attributes of $R_1(X_1)$ and $R_2(X_2)$, $R_2(X_2)$ and $R_3(X_3), \ldots,$ $R_{n-1}(X_{n-1})$ and $R_n(X_n)$ respectively. The $JD^*(X_1, X_2, \ldots, X_n)$ holds in R. And, since lossless joins are both associative and commutative (Aho et al 1979), every (2,3,...,n-1) combinations of $R_1(X_1), \ldots, R_n(X_n)$ are JDs in R. The number of such JDs is given by $1 + \sum_{r=1}^{n-2} {}^{n}C_{r}$ (Appendix I). There are no other JDs in R apart from those on the join attributes $\{Y_1, \ldots, Y_{n-1}\}$. This is the case because of the necessary and sufficient condition for a lossless join. For any two of the mappings represented by $R_i(X_i)$ and $R_j(X_j)$, if there is an attribute A_i common to both X_i and X_j, $A_i \subset Y_i$, or R(X&i not a natural join of R_i and R_j. Hence, the rules in (2) and (3) need only ensure that the common attributes in the successive joins are candidate keys.

There are eleven distinct combinations of two mappings of the form A----B and B----C into a schema R(A,B,C) as in section 3.4.3. Only three of the combinations have the join attribute B as a member of the set of candidate keys. Rule (2) consists of those three combinations. Rule (3) allows successive combination of mappings with relation schemas generated by rule (2). Only five of the distinct combinations have their join attributes as candidate keys; the rule consists of those combinations.

CHAPTER IV . Evaluation of the Full Mapping Approach

There are two aspects to this thesis. The full mapping as a means of expressing relationships between data items of a database is proposed. Transformation rules to generate relation schemas from the mappings are also presented. An assessment of the full mappings as an alternative to functional and multivalued dependencies, and a comparison of schema design via the transformation rules with other design methods are given in this chapter. In section 4.1, we run through a series of examples that show how anomalies are eliminated in the normalization process. The effect of full mapping and the transformation rules are discussed with the examples. A design example is given in section 4.3.

4.1 The Normalization Process

A relation is said to be in a particular normal form if it satisfies some constraints which are known to prevent certain update problems. The following examples from (Date 1980) illustrate the normalization process. The examples are based on a relation containing information about suppliers of machine components, the parts/quantity supplied and cities where suppliers are located.

4.1.1 The second Normal Form Problem

The information in the supplier-part environment can be represented as a table with no attribute-values repeated in the rows. Such a table can be described by a relation schema

FIRST(<u>SNO</u>, STATUS, CITY, <u>PNO</u>, QTY). The relation is said to be in first normal form. The funtional dependencies in FIRST are shown in figure 13.

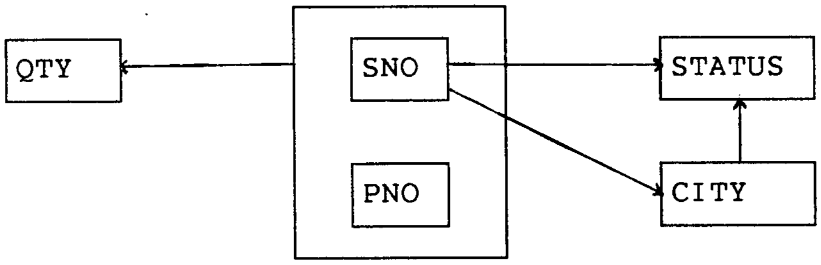

Figure 13. The FDs in Relation schema FIRST

The relation schema FIRST suffers from certain anomalies.

1.) Insertion: It is not possible to enter the fact that a particular supplier is located in a city until that supplier supplies at least one part. This is the case because no component of a primary key may be null.

2.) Deletion: A tuple containing a supplied part is deleted when the corresponding supplier no longer supplies that part. If the only tuple for a particular supplier is deleted, the information that the supplier resides in a city is destroyed.

3.) Redundancy: The city-value for a supplier appears in as many tuples as there are parts supplied by the supplier. The redundancy causes update search problems and gives room for potential inconsistencies.

A possible normalization solution is to decompose the relation schema FIRST into SECOND(<u>SNO</u>, STATUS, CITY) and SP(<u>SNO</u>, <u>PNO</u>, QTY). The solution eliminates the nonfull functional dependencies of STATUS and CITY on the key. The nonfull functional dependency problem is suffered by every relation that is not in second normal form (2NF). A relation is 2NF if and only if it is in first normal form and every nonkey attribute is fully dependent on the primary key.

A formal analysis of the supplier-part environment will reveal the following facts from which full mappings can be derived.

1.) The quantity of a part (QTY) is only meaningful when associated with a part and its supplier. Therefore, the association (PNO----SNO) is indivisible and should be treated as an entity.

2.) A supplier supplies many parts and a part may be supplied by many suppliers.

3.) A supplier is located in a city even when he currently supplies no parts. There may be some cities without suppliers.

The full mappings in the application environment are as follows.

SNO----CITY (many-one, onto - into)

CITY----STATUS (one-one, onto - onto)

SNO----PNO (many-many, into - onto)

(SNO, PNO)----QTY (many-one, onto - onto)

Some facts that are not revealed by the functional dependencies can now be expressed. The functional dependency of CITY on SNO is represented by the many-one mapping between SNO and CITY. But the onto/into status of the full mapping further states that there can be some cities within the database with no suppliers residing in them. The update problems in FIRST arise as a result of combining relations over an attribute involved in 'into' and 'onto' mappings. The schema SP(<u>SNO</u>, <u>PNO</u>, QTY) should never have been combined with (SNO----CITY).

4.1.2 The Third Normal Form problem

The relation schema SECOND(<u>SNO</u>, STATUS, CITY) still suffers from certain update problems.

1.) Insertion: It is not possible to enter (CITY, STATUS) value until there are some suppliers located in that city.

2.) Deletion: Similarly, if the only supplier in a city is deleted, the city/status information is lost.

3.) Redundancy: There is still some redundancy due to city/status value that is being repeated for as many suppliers in a city.

A normalization solution replaces the relation schema SECOND by SC(<u>SNO</u>, CITY) and CS(<u>CITY</u> , STATUS). There is a transitive dependence of STATUS on SNO. SECOND is not in third normal form. A relation is in third normal form (3NF) if and only if every nonkey attribute is nontransitively dependent on the primary key.

From the full mapping viewpoint, (SNO----CITY) and (CITY----STATUS) are not combinable. The join attribute CITY is mapped 'onto' in CITY----STATUS and 'into' in SNO----CITY. This in fact, explains the insertion/deletion problems in SECOND more than transitive dependency. The insertion and deletion anomalies will not occur if CITY is mapped 'onto' in both mappings.

4.1.3 The Boyce-Codd Normal Form Problem

The relation schema SCHOOL(<u>STUDENT,</u> <u>SUBJECT,</u> TEACHER) originally appeared in (Date 1980). The functional dependency diagram and a sample relation are given in figure 14 and 15 respectively. The following facts are true in the application

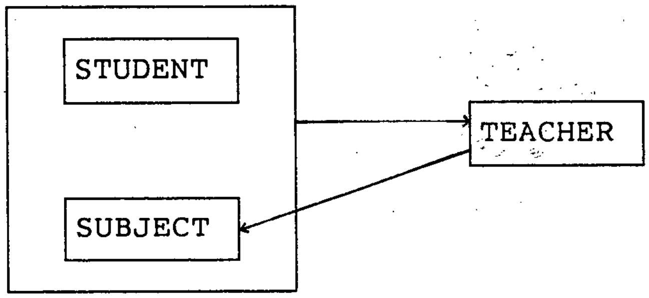

Figure 14. FDs in Relation Schema SCHOOL

environment.

1.) For every subject, a student is taught by only one teacher.

2.) A teacher teaches only one subject, but each subject is taught by several teachers.

STUDENT	SUBJECT	TEACHER
SMITH	MATH	Prof WHITE
SMITH	PHYSICS	Prof GREEN
JONES	MATH	Prof WHITE
JONES	PHYSICS	Prof BROWN

Figure 15. Sample Relation for schema SCHOOL

The relation SCHOOL is in 3NF, but it suffers from certain update problems. We can not delete such information as "Jones is studying physics" without losing the information that prof. Brown teaches physics. The problem arises from TEACHER being a determinant, but not a candidate key in the relation. This is the Boyce-Codd normal form problem. We recall that a relation R is in Boyce-Codd normal form if and only if every determinant is a candidate key in R.

A normalization solution decomposes the schema into ST(STUDENT, TEACHER) and TS(TEACHER, SUBJECT). Both ST and TS are in BCNF and the update problem is taken care of. But different problems have been introduced. The ST relation does not provide much useful information. The relationship between a student and a teacher is only meaningful with respect to subject.

The full mappings for the database are as follows:

SUBJECT^{o} m----n STUDENT^{o}

$(\text{SUBJECT, STUDENT})^{o}$ 1----1 TEACHER^{o}

TEACHER^{o} m----1 STUDENT^{o}

From the BCNF transformation rules, TS(TEACHER, SUBJECT) and SST(SUBJECT, STUDENT, TEACHER) are generated. The relation schema SST is atomic and can not be decomposed. It is also not possible to combine TS and SST. This case actually turns out to be an example where no amount of decomposition will produce the desired relations.

4.1.4 The fourth Normal Form Problem

A relation schema CTX(COURSE, TEACHER, TEXT) describes a situation where, for any given course, there may exist any number of corresponding teachers and texts. TEACHER and TEXT are assumed to be independent. That is, there are multivalued dependencies COURSE-->-->TEXT and COURSE-->-->TEACHER in CTX.

There are no functional dependencies. A sample relation is given in figure 16.

COURSE	TEACHER	TEXT
PHYSICS	Prof. GREEN	BASIC MECHANICS
PHYSICS	Prof. GREEN	PRINCIPLES OF OPTICS
PHYSICS	Prof. BROWN	BASIC MECHANICS
PHYSICS	Prof. BROWN	PRINCIPLES OF OPTICS
MATH	Prof. WHITE	MODERN ALGEBRA
MATH	Prof. WHITE	PROJECTIVE GEOMETRY

Figure 16 Sample Relation for CTX

The relation CTX contains a lot of redundancy. A new text for a course will require entries for every teacher that teaches the course. A solution is to decompose CTX into CT(COURSE, TEACHER) and CX(COURSE, TEXT) based on the multivalued dependencies of TEXT and TEACHER on COURSE. The problem is that CTX is not in fourth normal form.

A relation is in fourth normal form (4NF) if and only if whenever there exists an MVD in R, say A-->-->B, then all attributes of R are also functionally dependent on A.

The full mappings in CTX are TEACHER^{o} m----1 COURSE^{o} and TEXT^{o} m----1 COURSE^{o}.

If the two mappings are combined over COURSE, then a given course has to be repeated for all the teacher/text combinations. This is precisely what fourth normal form is to eliminate. Although we did not define rules for generating 4NF schemas, the PJ/NF rules will not allow a join of (TEACHER----COURSE) and (TEXT----COURSE). Projection-join normal form relations also conform to the fourth normal form (Fagin 1979).

4.1.5 The Projection-join normal form problem

We recall that a join dependency constraint JD*(X,Y,...,Z) holds in a relation R, if R is equivalent to the join of its projections over X,Y,...,Z where X,Y,...,Z are combinations of attributes of R. However, as illustrated in section 2.1, JD constraints are not easy to maintain. The relation SPJ in figure 7 suffers from a number of update problems due to its JD constraint. An insertion of a tuple may call for other tuples to be inserted. Similarly, a deletion may require that some other tuples be deleted.

However, not all JD constraints have the update maintenance problems. Relations with such problem-free JDs are said to be in projection-join normal form. A relation R is in projection-join normal form (PJ/NF) if and only if every join dependency in R is implied by a candidate key of R. A join dependency JD*(X,Y,...,Z) in R is implied by a candidate key of R if the join attributes in X,Y,...,Z uniquely determine tuples of R.

The problems in the relation SPJ arise because the join attributes S, P, and J are not keys. The full mappings for the relationships in SPJ are S^{o} m----n P^{o}, P^{o} m----n J^{o} and

$\overset{o}{S}$ m----n $\overset{o}{J}$. According to the PJ/NF transformation rules, the relations R(S, P), R(P, J) and R(S, J) are in their final form; they can not be combined in any way.

4.2 Full Mapping versus FDs and MVDs

Full mapping, as a means of expressing relationships between data items of a database, compares favorably with functional and multivalued dependencies. Two of the three basic mapping types can express functional dependency. Let X and Y be attributes representing some entity or property-value sets.

1.) One-one mapping X 1----1 Y expresses the functional dependency of Y on X and vice versa. That is, X--->Y and Y--->X. The corresponding atomic relation schema is either R($\underline{X}$, Y) or R(X, $\underline{Y}$).

2.) Many-one mapping X m----1 Y expresses the functional dependency of Y on X as well as the fact that X is not functionally dependent on Y. That is, X--->Y and Y-/->X. The corresponding atomic relation schema is R($\underline{X}$, Y).

The mappings explicitly specify functional relationships in both directions. The same amount of information can only be inferred from two or more functional dependencies. In addition, the into/onto status of the mappings provides some information that can not be expressed in functional dependency specifications. A mapping $\overset{i}{\text{NAME}}$ m----1 $\overset{o}{\text{PHONE}}$ in a company database expresses the functional dependency of PHONE on NAME. But in addition, it specifies the fact that,at a database instance, a name may have no phone associated with it.

Nonfunctional relationships can be expressed as many-many or many-one mapping. The many-one mapping, as shown above, specifies a functional dependency in one direction and nonfunctional relationship in the other. The many-many mapping specifies nonfunctional relationship in both directions: X-/->Y and Y-/->X. The corresponding relation schema is R($\underline{X}$, $\underline{Y}$).

The multivalued dependency, as a means of expressing nonfunctional relationships, is such that it is only recognizable when it coexists with another one in the same relation. While an FD X--->Y is defined only in terms of the sets X and Y, the validity of an MVD X-->-->Y in a relation R(U) depends on the values of all the attributes in U. The MVD can not be derived from R[XY].

Let X and Y be subsets of U, and let Z be the complement (in U) of the union XY. For any relation R(U), the multivalued dependency X-->-->Y holds in R if and only if R is the natural join of its projections R[XY] and R[XZ]. The MVD X-->-->Z also holds in R (Beeri et al 1977).

By definition, MVDs are not only unintuitive, but their properties are not well understood. An MVD may hold in a projection, but not in the parent relation. Such MVDs are said to be <u>embedded</u> . Some embedded MVDs can be obtained by projectability from the MVDs in the parent relation. The projectability rule states that if X-->-->Y holds in R(U) and X c Z $\underline{c}$ W, then X-->-->(Y n Z) holds in R[Z]. The MVDs that can not be derived are said to be <u>latent</u> in the relation (Zaniolo and Melkanoff 1981).

The many-many and many-one mappings are equivalent to trivial MVDs. An MVD X-->-->Y which holds in R(U) is trivial if U = X u Y or Y $\subseteq$ X. The relationships between the mappings and MVDs can be examined in relations derived from two or more mappings. But the only inference rule from projections to a join is the joinability rule which states that:

if 1.) R(W u Z) = S(W).P(Z)

2.) X-->-->Y holds in S(W), and

3.) Y n Z = Ø then X-->-->Y holds in R(W u Z) (Zaniolo and Melkanoff 1981).

The joinability rule, as defined above, only deals with cases where the attributes Y and Z are disjoint. And, as stated in section 2.3.2, two many-many mappings E m----n S and E m----n D will, in general, not appear as an MVD if there exists some relationships involving S and D (Kent 1981). Thus, because of the nature of multivalued dependencies, the relationship between them and mapping types is not quite clear. However both MVDs and full mappings model the many-many relationships, but some MVDs may be hidden in database relations. The mappings are such that they can be recovered by projection over the join attributes. The combination of two mappings X----Y and X----Z results in MVDs X-->-->Y|Z, if there are no other mappings between Y and Z in the same relation. The into/onto status of full mappings dictates which mappings can be combined. Hence, given all the mappings in an application environment, they can be expressed as trivial and nontrivial MVDs depending on which mappings are combined. The full mappings provide at least as much information as the multivalued dependencies.

4.3 Design Example

The design example in (Wong et al 1980) will be used to illustrate the transformation rules for generating relation schemas from full mappings. From the example, the following entity and property-value sets can be identified.

Entity Sets	Property-value Sets
EMPNO	ENAME, BIRTHYR
DEPTNO	DNAME, LOCATION
JOBNO	TITLE, SALARY

A property has a one-one or one-many correspondence with an entity set. Every entity has exactly one property and every element in a property-value set is associated with some entities. The value associations are as follows.

1.) EMPNO^{o} m----1 ENAME^{o}

2.) EMPNO^{o} m----1 BIRTHYR^{o}

3.) DEPTNO^{o} 1----1 DNAME^{o}

4.) DEPTNO^{o} m----1 LOCATION^{o}

5.) JOBNO^{o} m----1 TITLE^{o}

6.) JOBNO^{o} m----1 SALARY^{o}

There are six entity set associations.

1.) EMPNO^{o} m----1 DEPTNO^{o} derived from the association works-in(EMP,DEPT).

2.) EMPNO^{o} 1----1 JOBNO^{i} derived from assignment(EMP,JOB) association. It is assumed that a job may not be filled. The assumption is consistent with the example and the definition of association in (Wong et al 1980).

3.) DEPTNO^{o} 1----1 EMPNO^{i} derived from the mgr(DEPT,EMP) relationship. A department may have only one manager because the "mgr" relationship is single-valued.

4.) EMPNO^{o} m----n JOBNO^{o} derived from the general relationship qualified(EMP,JOB). Let us assume that every job has some qualified employee and that an employee is qualified for at least one job.

5.) DEPTNO^{o} m----n JOBNO^{o} derived from the relationship allocation(DEPT,JOB). A job may be allocated to more than one department and a department may have many jobs.

6.) (DEPTNO^{o} m----n JOBNO^{o})^{o} 1----1 NUMBER^{o} defines a property "number" on the allocation relationship.

Using the transformation rules, BCNF relation schemas can be generated from the mappings. All the mappings are in one common attribute group. They can be combined as follows.

1.) EMPNO^{o} m----1 ENAME^{o} + EMPNO^{o} m----1 BIRTHYR^{o}

+ EMPNO^{o} 1----1 JOBNO^{i} + EMPNO^{o} m----1 DEPTNO^{o}

2.) DEPTNO^{o} 1----1 DNAME^{o} + DEPTNO^{o} m----1 LOCATION^{o}

+ DEPTNO^{o} 1----1 EMPNO^{i}

3.) JOBNO^{o} 1----1 TITLE^{o} + JOBNO^{o} m----1 SALARY^{o}

4.) EMPNO^{o} m----n JOBNO^{o}

5.) (DEPTNO^{o} m----n JOBNO^{o}) 1----1 NUMBER^{o}

+ DEPTNO^{o} m----n JOBNO^{o}

The corresponding relation schemas and candidate keys are as follows.

1.) EMP(EMPNO, ENAME, BIRTHYR, DEPTNO, JOBNO)	{EMPNO, JOBNO}
2.) DEPT(DEPTNO, DNAME, LOCATION, EMPNO)	{DEPTNO, DNAME, EMPNO}
3.) JOB(JOBNO, TITLE, SALARY)	{JOBNO, TITLE}
4.) QUALIFIED(EMPNO, JOBNO)	{EMPNO-JOBNO}
5.) ALLOCATION(DEPTNO, JOBNO, NUMBER)	{DEPTNO-JOBNO}

4.4 Full Mapping Approach versus other Design Methods

This section compares the full mapping/transformation rules, as a design approach, with other methodologies discussed in chapter (2). The comparison has two sides to it. A comparison is made between full mappings and the inputs in other methods, as well as between the nature of the transformation rules and other design processes.

4.4.1 Synthetic Methods and the Full Mapping Approach

The full mapping approach to relational database design is also synthetic in the sense that the design process starts with a set of attributes and a statement of the relationships among them. An earlier synthesis algorithm (Bernstein 1976) uses the minimum cover technique. This involves a purely syntactic treatment of functional dependencies. The technique demands that the functional dependency between any two attributes be unique. But this is not always the case in practice, hence attributes may have to be renamed to maintain the uniqueness assumption. The design results, in turn, have to be validated semantically.

In the full mapping approach, every relationship is specified independent of others. Each mapping has an intension and can not be derived from other mappings. Consequently, two mappings with exactly the same specifications are not necessarily the same. The mapping X 1----1 Y and X m----1 Y together with their into/onto status provides six different ways of expressing a functional dependency between attributes X and Y. Thus, full mapping provides a discipline for dealing with the uniqueness assumption and the need for semantic validation of design results. The problems associated with uncontrolled renaming of attributes have been discussed in chapter (2). The into-mapping allows a relationship to be defined on a subset of a domain without having to rename the attribute.

The input to the synthesis algorithm is also limited to functional dependencies. Nonfunctional relationships are entered indirectly. The algorithm essentially generates a relation per each nonfunctional relationship. And because of the nature of multivalued dependency, the synthesis algorithms have not been extended to 4NF and PJ/NF designs.

The many-many and one-many mapping types describe nonfunctional relationships between data items. Thus, it is now possible to model nonfunctional relationships in a context-independent form, and to generate PJ/NF schemas via the transformation rules. From a practical viewpoint, the rules appear more desirable than the minimum cover technique. Design results need not be subjected to semantic validation. All semantic considerations take place at the mapping specification stage.

4.4.2 Decomposition and the Full Mapping Approach

The decomposition methods, in general, take as input functional and multivalued dependencies and an initial relation design. The MVD models nonfunctional relationships. However, as discussed in section 4.2, the properties of MVDs are not well understood. Multivalued dependencies are only valid within the context of a relation and can only be detected in relations. However, they can not be easily detected. There are other problems relating to embedded and latent dependencies. Different decomposition paths can lead to different designs of varying qualities. Choice of decomposition is mainly based on heuristics.

Some of these problems are dealt with extensively in (Zaniolo and Melkanoff 1981). It was noted that some dependencies are lost in a decomposition process. Hence, decomposition is based on complete data relatability to ensure that all the dependencies are preserved. The data relatability condition is such that the initial dependencies can be inferred from the projections. Only the paths that preserve the dependencies are used in decomposition. A limitation to this approach is brought about by the properties of MVDs. An MVD can only be inferred (from a projection to a join) using the joinability rule. It is not clear whether some other MVDs are inferable by stronger rules.

Most decomposition methods also assume an initial relation. This is referred to as the universal relation assumption (Kent 1981). The consequences of the universal relation assumption, as discussed in chapter (2), are not compatible with practical database design. The universal meaning for column names has associated with it an inter-relational constraint which is not easy to maintain. Decomposition methods deal with the column name problem by renaming attributes. But, as mentioned in chapter (2), uncontrolled renaming of attributes can create join problems in query processing.

The problems relating to MVDs can be avoided by using the full mappings to model nonfunctional relationships. Full mappings are easy to comprehend and they can express at least as much information as multivalued dependencies. And since the full mapping approach is synthetic in nature, assuming an initial relation and its consequences have been avoided.

A comparison can also be made from the point of view of synthesis versus decomposition. Decomposition appears superior to earlier synthetic methods because nonfunctional relationships can be modeled directly. The full mappings now provide a means of modeling nonfunctional relationships within a synthetic approach. The transformation rules are such that every relation schema generated can be projected back to the initial mappings. If it can be shown that the full mappings capture all the relationships we wish to express, the synthetic approach will clearly be superior to decomposition.

4.4.3 Entity-Relationship Approach and the Full Mapping Rules

The Entity-Relationship approach offers a practical method for database design. It takes as input entity sets, associations, relationships and properties (Wong et al 1980). The Full mapping approach can be grouped in the same category as the Entity-relationship method in the design method summary (figure 9); they share a lot in common.

The intermediate model in the entity-relationship approach can be specified as full mappings. But full mappings, through the into/onto status, allow more semantic information to be specified. Within the E-R approach, the single-valued binary relationships and the associations also specify implicitly some into/onto status information. A binary relationship R_t between entity sets $E_1(t)$ and $E_2(t)$ is single-valued if each entity occurs in at most one instance of R_t. That is, some entities in $E_1(t)$ may not participate in the relationship. An association is a binary relationship in which $E_1(t)$ entities occur in exactly one instance; an 'onto' mapping is specified on $E_1(t)$. Within the framework of full mappings, relationships and associations are treated uniformly. The into/onto status information is specified for all the sets involved. The properties have many-one or one-one association with the entity or relationship sets, and are always 'onto'. Thus, more semantic information can be expressed in the full mappings than the intermediate model of the E-R approach.

There are three rules for transforming the E-R intermediate design model into relation schemas. Rule (1) assigns an explicit identifier for each entity set. Rule (2) groups the identifier of a primitive object (an entity set or relationship) with all its properties or associations in the same relation.
However, it is possible to have two associations ($E_1(t)$, $E_2(t)$) and ($E_1(t)$, $E_3(t)$) such that at time t the extensions of $E_1(t)$ in the associations are not equal. The grouping of such associations together in a relation will prevent insertion of a tuple when the extensions are not equal. Hence rule (2) may generate relations with this type of insertion anomaly.
Rule (3) allows one and only one primitive object per relation. The rule is too restrictive. Two relationships can be grouped together in a relation as long as they have a set in common and the set appearing in the two relationships have same domain extensions. Thus, a violation of rule (3) does not necessarily result in an 'update anomaly' as defined in (Wong et al 1981).

Both the Entity-Relationship method and the Full mapping approach offer what seem like a practical design methodology. The basic difference between the two approaches is in the full mapping specification which allows more semantic information to be expressed. The additional information allows a certain kind of insertion/deletion anomaly to be avoided. It also allows more relaxed rules. The full mapping rules also take advantage of previous work on relational database theory.

CHAPTER V . Conclusions

5.1 Achievements

The full mapping approach to relational database design can be viewed in two ways: the full mappings as a means of specifying data dependency constraints, and the nature of the transformation rules. We have been able to incorporate more semantics into data dependency specification. The into/onto status information of a mapping, as discussed in earlier chapters, can specify certain information that can not be expressed in a functional dependency. The full mappings also compare favorably with multivalued dependency. Two nonfunctional relationships X----Y and X----Z are expressed as a MVD in a relation schema R(XYZ), if R(XYZ) = R[XY].R[XZ]. Implicitly, the MVD states that the extension of X in R(XY) is always equal to its extension in R(XZ). Within the full mapping approach, nonfunctional relationships are specified as mappings between two sets. The mappings can be represented by atomic relation schemas. The condition for a lossless join of the schemas, the equivalence of domain extensions of the join attributes, is derived from the into/onto status information. Thus the many-many relationship between any two data items can be expressed out of context of a relation. The full mappings, in comparison to MVDs, are simple and intuitive.

The earlier synthesis algorithms have not been extended to fourth normal form schemas because nonfunctional relationships can not be represented directly. Specifying nonfunctional relationships as MVDs necessarily requires an initial relation. The grave consequences of the Universal Relation assumption are discussed in (Kent 1981). But, it is now possible to model nonfunctional relationships out of context of a relation, and hence within a synthetic approach. Transformation rules are defined to generate BCNF and PJ/NF schemas from full mappings. Using a decomposition approach to design PJ/NF schemas will entail detecting the join dependencies in a set of initial relations. This is a tedious task and may not be practicable even in small databases. As noted in (Date 1980), the process of determining when a given relation is in 4NF but not PJ/NF (and hence can be decomposed) is still unclear.

Full mappings would provide a practical approach to automatic design of relational database schemas. As noted in (Tsichritzis and Lochovsky 1982), database theory is more of a schema analysis than schema design. The theory provides deeper understanding of the data models, the database schemas, and their properties. But it is not readily applicable. It should be treated as a tool for understanding, and not necessarily as a tool for design. We believe that the full mapping approach lends itself to practical database design. From an information analysis of an application environment, the relationships among the data items can be represented as full mappings. Relation schemas are generated via the transformation rules.

The are no rules defined for 4NF schemas, because we have not formally stated the relationships between the mappings and multivalued dependency. Although some insights into understanding MVDs have been gained, the relationship between the two is not quite clear. Two many-many relationships E m----n S and E m----n D in a schema R(ESD) are expressed as a JD*(ED, ES, SD), rather than as an MVD, if there is a many-many relationship between S and D. The demarcation between MVDs and JDs is not very clear. However, transformation rules are defined for PJ/NF schemas. Projection-join normal form implies fourth normal form (Fagin 1979).

The sets of transformation rules not only produce relation schemas that conform to their respective normal forms, but they also eliminate certain anomalies that may exist in normal form schemas. For example, let R($\underline{A}$,B,C) be a relation schema. If the only dependencies in R are A--->B and A--->C, then it is in Boyce-Codd normal form. However, the relationships among A, B and C may be such that A^{i} m----1 B and A^{o} m----1 C. The mappings express the functional dependencies, but in addition A is mapped 'into' in one of the mappings. Therefore, at a database instance when the extensions of A in the two mappings are not equal, it will not be possible to enter a certain tuple without null values. This type of insertion/deletion anomalies can still be present in BCNF relations. The BCNF transformation rules eliminate such anomalies by combining only over attributes with equivalent domain extensions.

The full mapping approach also provides a discipline for dealing with some of the problems related to renaming of attributes and the universal meaning of column names. The domains of attributes are also given a consideration within the design approach. The into-mapping allows an association to be defined on a subset of a domain without having to rename the corresponding column name. Thus, attributes or column names do not necessarily have a universal meaning. The defined sets and mappings do have intensions. From the intensions, it should be clear which domains have equivalent extensions. Hence, by referring to the statement of intentions during query processing, joins would be carried out only over those domains that ensure the lossless property. However, it is important that we are able to keep track of the sets, the mappings and their intensions. The topic is not considered in this thesis. More work is needed in this area. It is hoped that the full mappings would serve as a basis for the kind of generalized interdependency constraint specification envisaged in (Kent 1981).

5.2 Further Research

As noted above, further work is required on the integrated data dictionary. This would involve organizing the sets, the mappings and the statement of intensions into a structure that can be managed by the Database Management System in use. The dictionary would serve as a kernel, from which the database can be designed. A related problem is the automation of database design. From a database kernel, it should be possible to derive database schemas automatically. The computational problems relating to the transformation rules may have to be studied.

Another possible research is to reformalize the relational database theories using the full mapping approach. If the relationships between full mappings and multivalued dependencies can be formally stated, schema analysis might be better understood from the full mapping viewpoint.

Naturally, this work should be extended to the Network and Hierarchical database design. Transformation rules could be defined to generate Network and Hiera schemas from full mappings. The work may lead to a comprehensive and automated design system. If the system is able to generate schemas for the three conventional models, it might serve as a basis for testing the suitability of the data models for different application environments.

Bibliography

Aho, A. V., Beeri, C., and Ullman, J. D. "The Theory of Joins in Relational Databases," ACM Transactions on Database Systems , Vol. 4, No. 3, Sept. 1979.

Armstrong, W. W. "Dependency Structures of Database Relationships," Proc. Int'l Federation for Information processing Congress , North Holland, 1974.

Beeri, C., and Bernstein, P. A. "Computational Problems Related to the Design of Normal Form Relational Schemas," ACM Transactions on Database Systems , Vol. 4, No. 1, March. 1979.

Beeri, C., Bernstein, P. A., and Goodman. N. "A Sophisticate's Introduction to Database Normalization Theory," . Proc. 4th Int'l Conference on Very Large Databases , Berlin, 1978.

Beeri, C., Fagin, R., and Howard, J. H. "A Complete Axiomatization for Functional and Multivalued Dependencies in Database relations," Proc. ACM SIGMOD Conference , Toronto, 1977.

Bernstein, P. A. "Synthesizing Third Normal Form Relations from Functional Dependencies," ACM Transactions on Database Systems , Vol. 1, No. 4, Dec. 1976.

Bernstein, P. A., Swenson. J. R., and Tsichritzis, D. C. "A Unified Approach to Functional Dependencies and Relations," Proc. ACM SIGMOD Conference , San Jose, 1975.

Biskup, J., Dayal, U., and Bernstein, P. A. "Sythesizing Independent Database Schemas," Proc. ACM SIGMOD Conference , May 1979.

Cadiou, J. M. "On Semantic Issues in the Relational Model of Data," Proc. Int'l Symposium of Mathematical Foundations of Computer Science , Poland, Sept 1975.

Codd, E. F. "A Relational Model for Large Shared Data Bases," Communications of the ACM , Vol. 13, No. 6, June 1970.

Codd, E. F. "Further Normalization of the Data Base Relational Model," Courant Computer Science Symposium 6, Data Base Systems , Prentice-Hall, May 1971.

Codd, E. F. "Extending the Database Relational Model to Capture More Meaning," ACM Transactions on Database Systems , Vol. 4, No. 4, Dec. 1979.

Chen, P. P. "The Entity-Relationship Model: Toward a Unified View of Data," ACM Transactions on Database Systems , Vol. 1, No. 1, March 1976.

Date, C. J., Introduction to Database Systems, 3rd ed., Addison-Wesley,Reading, Mass, 1980.

Dayal, U., and Bernstein, P. A. "The Updatability of Relational Views," Proc. 4th Int'l Conference on Very Large Databases , Berlin, 1978.

Delobel, C. "Normalization and Hierarchical Dependencies in Relational Data Model," ACM Transactions on Database Systems , Vol. 3, No. 3, Sept. 1978.

Delobel, C., and Casey, R. G. "Decomposition of a Database and the Theory of Boolean Switching Functions," IBM Journal of Research and Development , Vol. 17, No. 5, Sept. 1972.

Delobel, C., and Léonard, M. "The Decomposition Process in a Relational Model," Proc. Int'l Workshop on Data Structure Models for Information Systems , Belgium, May 1974.

Delobel, C., and Parker, D. S. "Functional and Multivalued Dependencies in Relational Database and the Theory of Boolean Switching Functions," Tech. Report No. 142, Dept. Maths Appl. et Informatique , Univ. de Grenoble, France. Nov. 1978.

Ehrig, H., Kreowski, H. J., and Weber, H. "Algebraic Specification Schemas for Database Systems," Proc. 4th Int'l Conference on Very Large Databases , Berlin, 1978.

Fagin, R. "Multivalued Dependencies and a New Normal Form for Relational Databases," ACM Transactions on Database Systems , Vol. 2, No. 3, Sept. 1977.

Fagin, R. "The Decomposition Versus the Synthetic Approach to Relational Database Design," Proc. Int'l Conference on Very Large Databases , Oct. 1977.

Fagin, R. "Normal Forms and Relational Database Operators," Proc. ACM SIGMOD Conference , May 1979.

Fagin, R. "Horn Clauses and Database Dependencies," Proc. 12th Annual ACM Symposium on Theory of Computing , Los Angeles, April 1980.

Fagin, R. "A Normal Form for Relational Databases That is Based on Domains and keys," ACM Transactions on Database Systems , Vol. 6, No. 3, Sept. 1981.

Gerritsen, R. "Tool for the Automation of Database Design," NYU Symposium on Database Design , May 1978.

Haseman, W. D., and So, Y. H. "An Integrative Approach to Database Design," Working Paper, Carnegie Mellon University , Dec. 1977.

Housel, B.C., Waddle, V., and Yao, S. B. "The Functional Dependency Model for Logical Database Design," Proc. 5th Int'l Conference on Very Large Databases , 1979.

Kahn, B. K. "A Structured Logical Design Methodology," NYU Symposium on Database Design , May 1978.

Kent, W. "Consequences of Assuming a Universal Relation," ACM Transactions on Database Systems , Vol. 6, No. 4, Dec. 1981.

Lien, Y. E. "Multivalued Dependencies With Null Values in Relational Databases," Proc. 5th Int'l Conference on Very Large Databases , 1979.

Ling, T., Tompa, F. W., and Kameda, T. "An Improved Third Normal Form for Relational Databases," ACM Transactions on Database Systems , Vol. 6, No. 2, June 1981.

Nicolas, J. M. "Multivalued Dependencies and Some Results on Undecomposable Relations," Proc. 4th Int'l Conference on Very Large Databases , Berlin, 1978.

Parker, D. S., and Delobel, C. "Algorithmic Applications for a New Result on Multivalued Dependencies," Proc. 5th Int'l Conference on Very Large Databases , 1979.

Raver, N., and Hubbard, G. U. "Automated Logical Database Design: Concepts and Applications," IBM Journal of Research and Development , Vol. 16, No. 3, 1977.

Rissanen, J. "Independent Components of Relations," ACM Transactions on Database Systems , Vol. 2, No. 4, Dec. 1977.

Shipman, D. "The Functional Data Model and the Data Language DAPLEX," ACM Transactions on Database Systems , Vol. 6, No. 1, March 1981.

Smith, J. M., and Smith D. C. P. "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems , Vol. 2, No. 2, June 1977.

Tsichritzis, D. C., and Lochovsky, F. H., Data Models , Prentice-Hall, Englewood Cliffs, 1982.

Van de Riet, R. P. "On Multivalued Dependencies and Independencies," IBM Research Report RJ2380 , IBM San Jose Research Lab., 1978,

Wong, E. and Katz, R. H. "Logical Design and Schema Conversion for Relational and DBTG Databases," Entity-Relationship Approach to Systems Analysis and Design , (Chen, P. P. ed.), North Holland, 1980.

Zaniolo, C., and Melkanoff, M. A. "On the Design of Relational Database Schemata," ACM Transactions on Database Systems ,

Vol. 6, No. 1, March 1981.

Appendix I

Let $R(X_1 \cup X_2 \cup \ldots \cup Xn)$ be a relation schema resulting from some n mappings according to the transformation rules. If the corresponding atomic schemas representing the mappings are $R(X_1),\ldots,R(Xn)$, the $JD^*(X_1,X_2,\ldots,Xn)$ holds in R. The total number of join dependencies based on the join attributes in $JD^*(X_1,X_2,\ldots,Xn)$ is given by $1 + \sum_{r=1}^{n-2} {}^{n}C_{r}$.

The combination of n mappings, using the transformation rules, is such that the resulting relation schema is a lossless join of the atomic schemas denoting the mappings. That is, $R = R(X_1).R(X_2).\ldots.R(Xn-1).R(Xn)$. Let $\{Y_1,Y_2,\ldots,Yn-1\}$ be the join attributes of $R(X_1)$ and $R(X_2)$, $R(X_2)$ and $R(X_3),\ldots,$ $R(Xn-1)$ and $R(Xn)$ respectively. Since lossless joins are both associative and commutative (Aho et al 1979), every (2, 3,...,n-1) combinations of $R(X_1)$, $R(X_2)$, ..., $R(Xn)$ produces a JD in R, over the Yi's.

Let us refer to a $JD^*(Z_1,Z_2,\ldots,Zn)$ as an n-component join dependency. The total number of JDs, based on the join attributes Yi's, is given by the sum of the number of 2-component, 3-component, ..., and n-component dependencies. The 2-component dependencies are derived by factoring out the R(Xi)'s from $(R(X_1).R(X_2).\ldots.R(Xn))$. For every R(Xi), the corresponding join is $R(Xi).[R(X_1).\ldots.R(Xi-1).R(Xi+1).\ldots.R(Xn)]$. If $X' = X_1 \cup \ldots \cup Xi-1 \cup Xi+1 \cup \ldots \cup Xn$, the join produces a $JD^*(Xi, X')$. The number of such JDs is the same as the number of ways an item can be chosen from n items; it is given by $^{n}C_{1}$. Similarly, the number of r-component JDs is the same as the number of ways of choosing (r-1) items from n items where ordering is immaterial. The number is given by $^{n}C_{r-1}$ ($r = 3,\ldots,n-1$). The different ways of factoring out (n-1) Ri's produces the same join dependency. JD*($X_1,X_2,\ldots,Xn$) is the only n-component dependency. Other combinations do not produce JDs that are distinct from JD*($X_1,X_2,\ldots,Xn$), because of the associative and commutative properties of lossless joins. Hence, the total number of JDs over the Yi's is given by $1 + \sum_{r=1}^{n-2} {}^{n}C_{r}$.