

**THEORY OF NON-FIRST NORMAL FORM
RELATIONAL DATABASES**



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

**APPROVED BY
SUPERVISORY COMMITTEE:**

Ray Hunt

Don Batoy

A. Selberschuch

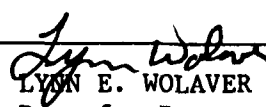
E. M. Emme

J. C. Brown

AD-A171 285

1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 86-120D	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Theory Of Non-First Normal Form Relational Databases		5. TYPE OF REPORT & PERIOD COVERED thesis /DISSERTATION
7. AUTHOR(s) Mark Aron Roth		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: The University of Texas		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433-6583		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 1986
		13. NUMBER OF PAGES 269
		15. SECURITY CLASS. (of this report) UNCLAS
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) B		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1		 LYNN E. WOLAVER 8 Aug 86 Dean for Research and Professional Development AFIT/NR
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED. DTIC FILE COPY		

-1-

20

Title: Theory of Non-First Normal Form Relational Databases

Name, Rank, Service: Mark Aron Roth, Capt, USAF

Degree, Year: Doctor of Philosophy, 1986

Institution: The University of Texas at Austin, Austin, Texas

Pages: 269

Abstract

One of the primary assumptions used in the relational model is that all relations must be in first normal form; that is, all values must be non-decomposable units. This assumption unduly constrains our ability to model data, especially for the non-traditional applications which are taxing our current database systems. This research extends relational database theory by relaxing the assumption that all relations in the database must be in first normal form. Relations containing attributes which may be atomic-valued or relation-valued are said to be in non-first normal form (non-1NF). In this context, we develop a non-1NF model and an extended formal query language based on the relational calculus, and prove its equivalence to a relational algebra extended with nest and unnest operators to deal with non-1NF relations. We define a property which non-1NF relations should satisfy, called partitioned normal form (PNF), and develop a set of extended algebra operators to manipulate non-1NF relations and maintain the PNF property. Our model and the extended operators are then further extended to deal with null values and empty nested relations. We present a user-oriented non-1NF query language, called SQL/NF, which is based on the SQL commercial database language and a proposed relational database language standard. Finally, we present a method for achieving nested normal form, a form which eliminates anomalies due to partial and transitive dependencies in PNF relations, and differs from previous algorithms by building non-1NF relations from an initial fourth normal form decomposition, incorporating embedded multivalued dependencies into the design, and improving upon the use of functional dependencies.

Acknowledgments

Thanks and praise are not enough to honor the Lord, Jesus Christ, who stood by me, encouraged me, and provided me with a multitude of friends and colleagues to help me complete this dissertation. First and foremost among those friends is my advisor Hank Korth. Without his guidance and untiring effort, this dissertation would not have been possible. I thank also the members of my committee, especially Avi Silberschatz and Don Batory, for their critical review of my work and helpful suggestions. Several conversations and meetings with Meral Özsoyoğlu of Case Western Reserve University gave me several ideas and clarified my objectives. In addition François Bancilhon, Carlo Zaniolo, Dirk Van Gucht, and several anonymous referees provided input at critical points in my research. Finally, I thank Fletcher Mattox and Arthur Keller for their help in the use of \TeX , which was used to typeset this dissertation.

Abstract

The advent of sophisticated software tools running on low-cost, powerful computers has prodded the database community into moving beyond the traditional data processing applications for which database systems were originally designed. Office forms, computer-aided design, and statistical database systems are but a few of the new applications for database systems which require new approaches to the database design and implementation. The foremost model for database use in the last decade has been the relational model. One of the primary assumptions used in the relational model is that all relations must be in first normal form; that is, all values must be non-decomposable units. This assumption unduly constrains our ability to model data, especially for the non-traditional applications which are taxing our current database systems. This research extends relational database theory by relaxing the assumption that all relations in the database must be in first normal form. Relations containing attributes which may be atomic-valued or relation-valued are said to be in non-first normal form (non-1NF). In this context, we develop a non-1NF model and an extended formal query language based on the relational calculus, and prove its equivalence to a relational algebra extended with nest and unnest operators to deal with non-1NF relations. We define a property which non-1NF relations should satisfy, called partitioned normal form (PNF), and develop a set of extended algebra operators to manipulate non-1NF relations and maintain

the PNF property. Our model and the extended operators are then further extended to deal with null values and empty nested relations. We present a user-oriented non-1NF query language, called SQL/NF, which is based on the SQL commercial database language and a proposed relational database language standard. Finally, we present a method for achieving nested normal form, a form which eliminates anomalies due to partial and transitive dependencies in PNF relations, and differs from previous algorithms by building non-1NF relations from an initial fourth normal form decomposition, incorporating embedded multivalued dependencies into the design, and improving upon the use of functional dependencies.

CONTENTS

Acknowledgements	iv
Abstract	v
Contents	vii
1 Introduction	1
1.1 Motivation	1
1.2 The State of \neg 1NF Research	4
1.3 Overview	6
1.4 Sequence of Presentation	11
2 The 1NF Relational Model	13
2.1 Basic Definitions	13
2.2 Relational Calculus	15
2.3 Relational Algebra	18
2.4 Data Dependencies	21
2.5 Normal Forms	26
2.6 Null Values	28
3 The \neg1NF Relational Model	30
3.1 \neg 1NF Data Models	30
3.2 Formal Query Languages for \neg 1NF Relations	36
3.3 Dependencies for \neg 1NF Relations	38
3.3.1 New Dependencies for \neg 1NF Relations	38

3.3.2 Using Dependencies on 1NF Relations	40
3.4 Normal Forms for \neg 1NF Relations	43
3.4.1 Horizontal Decomposition	43
3.4.2 Nested Normal Form	45
3.5 \neg 1NF Applications	48
3.5.1 Office Forms	48
3.5.2 Complex Objects and CAD	49
3.5.3 Statistical Databases	53
3.5.4 Relational Operating System Interface	53
3.5.5 Information Retrieval Systems	55
4 Formal Query Languages	57
4.1 Extended Relational Calculus	57
4.2 Extended Relational Algebra	62
5 Partitioned Normal Form and Extended Algebra Operators	67
5.1 Restricting the Class of \neg 1NF Relations	67
5.2 Extending the Basic Relational Algebra Operators	73
5.2.1 Extended Union	76
5.2.2 Extended Intersection	81
5.2.3 Extended Difference	84
5.2.4 Cartesian Product and Select	88
5.2.5 Extended Natural Join	88
5.2.6 Extended Projection	91

5.3 Closure of PNF Relations Under the Extended Operators	93
6 Equivalence of the Relational Calculus and the Relational Algebra	98
6.3.1 Reduction of Relational Algebra to Relational Calculus	98
6.3.2 Reduction of Relational Calculus to Relational Algebra	99
6.3.3 Examples	112
7 Null Values in \neg1NF Relational Model	116
7.1 Null Values in 1NF Relations	119
7.1.1 Basic Concepts	120
7.1.2 Operators	124
7.2 Introducing Null Values into the \neg 1NF Relational Model	138
7.2.1 Basic Concepts	140
7.2.2 Operators for \neg 1NF Relations with Nulls	143
7.2.3 Null-extended Operators	150
7.3 Dependencies in a Database with Null Values	159
7.3.1 Non-existent Nulls	160
7.3.2 Unknown Nulls	162
7.3.3 No-information Nulls	163
7.3.4 Join Dependency	164
8 The SQL/NF Query Language	165
8.1 Query Facilities	167
8.1.1 Nested Expressions	169

8.1.2 Functions	174
8.1.3 Null Values and Operations Dealing with Nulls	178
8.1.4 Miscellaneous Features	181
8.1.5 Data and Relation Restructuring Operations	183
8.1.6 Name Inheritance and Aliasing	186
8.2 Data Manipulation Language	188
8.3 The SQL/NF Data-Definition Language	193
8.4 Comparison with Other Languages	196
8.4.1 Generalized SQL	198
8.4.2 Query Language for NF ² Relations	199
9 A New Approach to Nested Normal Form	201
9.1 Definitions and Basic Procedures	208
9.2 The NNF Design Algorithm	217
9.3 Correctness of Algorithm 2	218
9.4 Further Normalization of NNF Relations	222
10 Conclusion	227
10.1 Summary of Results	227
10.2 Directions for Future Work	232
10.2.1 Recursive Schemes	232
10.2.2 Recursive Algebra and Optimization	233
10.2.3 Implementation	233
Appendix A	235

Appendix B	238
Bibliography	242
Vita	

Chapter 1

Introduction

1.1 Motivation

There has been a flurry of activity in recent years in the development of databases to support "high-level" data structures and complex objects. Office forms [SLTC], computer-aided design [BaKi], information retrieval systems [Sch1, SP], and statistical databases [OO2] are a few examples of non-traditional applications that require specialized database support. One of the stumbling blocks in using traditional relational databases is the assumption that all relations are required to be in first normal form (1NF); that is, all values in the database are non-decomposable. The 1NF assumption was a valid one for many years since database systems were geared to traditional business data processing tasks such as payroll and inventory. First normal form allowed a close mapping from physical data files to relations and simplified the theoretical and implementation problems for a model that was inefficient and slow to be accepted in early systems. Using today's computer technology, these reasons for relying on 1NF are not as valid as they once were. We now have the resources to deal with the type of applications mentioned above without constraining our databases by insisting on adherence to the 1NF restriction.

For this reason, non-first normal form (\neg 1NF) relations were proposed

in which the attributes of the relation can take on values which are sets or even relations themselves. Complex objects and semantically connected groups of data can more easily be represented as sets of values. For example, an object identifier is related to a set of object characteristics. This is more natural than forcing the view that we have many object identifier, object characteristic pairs as we would have to when sets are not available. This new assumption about sets in relations created a need to reexamine the fundamentals of relational database theory and opened the door for the introduction of new relational operators which take advantage of the *nested* structure of \neg 1NF relations.

To illustrate this, consider an employee relation in 1NF (Figure 1-1a), and a possible \neg 1NF structuring of it (Figure 1-1b). The \neg 1NF relation has two tuples,

$$\langle \text{Smith}, \{ \langle \text{Sam} \rangle, \langle \text{Sue} \rangle \}, \{ \langle \text{typing} \rangle, \langle \text{filing} \rangle \} \rangle$$

and

$$\langle \text{Jones}, \{ \langle \text{Joe} \rangle, \langle \text{Mike} \rangle \}, \{ \langle \text{typing} \rangle, \langle \text{dictation} \rangle, \langle \text{data entry} \rangle \} \rangle.$$

The Children and Skills attributes are *nested relations* each with one attribute, child and skill, respectively. The \neg 1NF relation makes clearer the independent associations of employee and skill, and employee and child, and reduces the data redundancy when compared with an equivalent 1NF relation.

An additional advantage of using a \neg 1NF structuring of the database is that fewer relations are needed to design a database that minimizes

employee	child	skill
Smith	Sam	typing
Smith	Sue	typing
Smith	Sam	filing
Smith	Sue	filing
Jones	Joe	typing
Jones	Mike	typing
Jones	Joe	dictation
Jones	Mike	dictation
Jones	Joe	data entry
Jones	Mike	data entry

(a)

employee	Children	Skills
	child	skill
Smith	Sam	typing
	Sue	filing
Jones	Joe	typing
	Mike	dictation
		data entry

(b)

Figure 1-1. Employee relation in (a) 1NF and (b) $\bar{1}$ NF.

redundancy, the cause of several undesirable properties. To illustrate this, consider that in the employee example of Figure 1-1a, we would expect that the employee-child relationship is independent of the employee-skill relationship. Thus, if we were to add a new child for employee "Jones," we would have to add three tuples, one for each skill that Jones currently has. Also, if we needed to change Smith's "typing" skill to "word processing" we would need to update a tuple for every child of Smith. In order to reduce redundancy and avoid update anomalies we would decompose this relation into its projections (employee, child) and (employee, skill). This result is shown in Figure 1-2. To view the entire database we must use a join operator to reassemble the original relation. The $\bar{1}$ NF relation of Figure 1-1b is free of the problems associated with the 1NF relation and uses one relation rather than the two required in the proposed decomposition. Furthermore, queries involving child and skill will be simpler and more efficient since a join of decomposed relations is not necessary.

employee	child
Smith	Sam
Smith	Sue
Jones	Joe
Jones	Mike

employee	skill
Smith	typing
Smith	filing
Jones	typing
Jones	dictation
Jones	data entry

Figure 1-2. Decomposition of employee relation of Figure 1-1a.

1.2 The State of \neg 1NF Research

A significant amount of research in the area of \neg 1NF relations has been done since the idea was first proposed in [Mak]. Most of this work has been published in the last four years and has been concentrated in the following four areas:

- (1) Development of data models to handle the new structure of \neg 1NF relations.
- (2) Development of a relational algebra for \neg 1NF relations and the various properties of such an algebra.
- (3) Exploration of new data dependencies which characterize \neg 1NF relations.
- (4) Introduction of a new normal form for nested relations with goals similar to traditional normalization techniques.

In addition, the first three of these areas developed in approximately three stages. At first, nested relations were limited to single attributes and only one level of nesting was allowed. This is the type of relation shown in Figure

1-1b. Then, the theory was generalized to many attributes and single levels, and finally to many attributes and multi-level nesting. The particulars of this previous work are summarized, for the most part, in Chapter 3, following an introduction to the 1NF relational model in Chapter 2. Where appropriate, we group related previous work with the chapter describing that area of new results.

Notably missing from this body of work was the development of a relational calculus for \neg 1NF relations. A calculus is a more formal language than an algebra and more clearly delineates the expressive power of the database language for a particular model. Our first task was to develop a \neg 1NF relational calculus and prove its equivalence to the \neg 1NF relational algebra. This work was reported in [RKS1] and is presented here in Chapters 4-6.

Another significant area where results had not been developed at all was in the addition of null values to the \neg 1NF model. The traditional null values could still be allowed for single valued attributes, but there was no attempt to deal adequately with the problem of null values for nested relations. Empty nested relations are a kind of null value and cause problems in query languages, if not carefully defined. An overview of research on null values and our extensions of the \neg 1NF model to include null values and empty nested relations were reported in [RKS2] and are presented here in Chapter 7.

We have also explored several areas related to database languages

and normalization. These include extended relational algebra operators, with and without null values in the model, a user-oriented language based on SQL (also described in [RKB]), and a new method for achieving normalized \neg 1NF relations. These areas are summarized in the next section.

1.3 Overview

In this section we present an overview of our contribution to the area of \neg 1NF relational database research. A major portion of this research is concerned with the development of query languages, both formal and user-oriented, for \neg 1NF databases. We define an extended relational calculus as the theoretical basis for our \neg 1NF database query language. We define an extended relational algebra and prove its equivalence to the extended calculus. In addition to the standard algebra operators, the extended algebra includes new operators, *nest* and *unnest*, first described by Jaeschke and Schek [JS], for manipulating nested relations. The *nest* operator forms nested relations by partitioning a relation based on the values of some attributes and collecting all tuples on the attributes being nested into a nested relation for each partition. The *unnest* operator eliminates a nested relation, concatenating each tuple of the nested relation with the other attributes of the relation.

Formal languages are useful for defining the retrieval power of the database language and as a basis for query optimization and implementation strategies. However, they are generally not used at the user level of database

interaction. Therefore, we define a \neg 1NF user language, called SQL/NF, which is based on the widely used commercial language, SQL [C+]. SQL/NF is designed using several important language design criteria [Dat2] and proposes a simpler structure than a concurrently developed language for \neg 1NF databases, undertaken at IBM Heidelberg [PT]. The SQL/NF language has all of the power of the extended relational calculus and algebra, adds capabilities for aggregate and other functions of the data, and adds language facilities for dealing with null values.

Null values require also a careful analysis in a more formal setting. Since we have the ability to represent multiple relationships in a single \neg 1NF relation without the problems of redundancy that doing so in a 1NF relation would entail, we must also deal with the fact that one or more of those relationships may be unknown or non-existent at some time. This means that we must deal with null values and their particular manifestation as empty nested relations in the \neg 1NF model. Thus, we look at a formalism for null values in the 1NF setting and extend those results to the \neg 1NF setting. We look at the *unknown*, *non-existent*, and *no-information* interpretation of nulls and how the empty nested relation fits into this framework. We use an open world assumption, where we assume that not only do relations contain the known information about the world, but that other information may belong there as well. To handle the different types of nulls we create a lattice of information based on the concept of *more informative*, with nothing more informative than

a known value or a non-existent null value and nothing less informative than a no-information null value. Using these concepts, we show that previous results [AM, Lie2] on the axiomatization of functional and multivalued dependencies in the presence of null values are incorrect. By using flawed reasoning concerning the inequality of non-existent nulls and the set of actual values which can replace unknown and no-information nulls, these authors modify the usual axiomatization of functional and multivalued dependencies into a much weaker one. We show that the usual axiomatization holds even in the presence of null values.

We are interested also in the design of $\neg 1NF$ databases. There are two classes of nested relations, based on the correspondence to their unnested counterparts. Some nested relations cannot be created from the corresponding unnested relation by any sequence of nest operators. An example is shown in Figure 1-3. Note that Smith and Jones are not single partitions and that there are two (Jones, typing) relationships, one of which would be eliminated in an unnested version of this relation. Relations of this class have the annoying property that there is not always a nest operation which will be an inverse for an unnest operation. There is also no semantic justification for allowing these relations. The relationship depicted in Figure 1-3, is that of employee and skill. There is no reason why each employee's skills should not all appear in one nested relation for that employee. If different sets of skills are related to employees in more than one way, then an additional attribute should be added to distinguish

employee	Skills
	skill
Smith	typing
	filing
Smith	sorting
	mailing
Jones	typing
	dictation
	data entry
Jones	sorting
	typing

Figure 1-3. $\neg 1NF$ relation which can not be achieved using the nest operation.

them. For example, the different sets of skills could represent different year's data, and so a "year" attribute should be added to the relation. Now employee and year will jointly identify skills sets, and nest will be an inverse for unnest. Therefore, we define a class of $\neg 1NF$ relations having the property that the atomic attributes of each relation and nested relation are a key for the relation. This property is called *partitioned normal form* (PNF). The PNF property is semantically equivalent to the structuring of $\neg 1NF$ relations via the *scheme trees* of [OY1] or the *formats* of [AB1].

By restricting the class of $\neg 1NF$ relations to those that satisfy the PNF property, we are able to provide some interesting extensions to the $\neg 1NF$ algebra operators. These new operators were inspired by the extended operators of [AB2], and have the property that the class of PNF relations is closed under their operation. They also maintain the implied multivalued dependen-

cies that exist in the 1NF counterparts of the operand relations, and so have better semantic underpinning for \neg 1NF relations than the standard relational operators. We define also versions of these operators for the \neg 1NF model which includes null values, and prove several equivalences among the operators for use in query optimization.

Although the class of PNF relations has certain desirable properties, there are further normalization techniques which can be applied to \neg 1NF relations. Some standards for normalization were proposed by [Lie1], but the most comprehensive approach to the problem has been done by Özsoyoğlu and Yuan [OY1]. They define *nested normal form* (NNF) which structures \neg 1NF relations based on the functional and multivalued dependencies which must exist in a 1NF database. There is a decomposition approach which breaks down the universe of attributes into a scheme tree and then splits off other scheme trees when partial or transitive dependencies exist. We provide a way of achieving NNF using a combination synthesis and decomposition approach which starts with a standard decomposition of the universe of attributes into fourth normal form (a "good" design for 1NF databases [Fag2]), employs given embedded multivalued dependencies to improve this decomposition, and then builds the scheme trees from this set of schemes. Our approach improves the design also by using functional dependencies in a more meaningful way. In [OY1], only the multivalued dependencies which are implied by the given functional dependencies are used in the design, thereby ignoring the different semantics of the

functional dependencies. By allowing the use of embedded multivalued dependencies and better utilizing functional dependencies, our approach can produce superior \neg 1NF schemes over the decomposition approach.

1.4 Sequence of Presentation

The remainder of this dissertation is organized as follows. Chapter 2 summarizes the 1NF relational model, providing the basic definitions upon which the \neg 1NF relational model is based. Chapter 3 presents the basic definitions of our \neg 1NF model, and discusses previous work in the areas of formal query languages, data dependencies, normal forms, and applications. Chapter 4 presents equivalent formal languages for the \neg 1NF model, including an extended relational calculus and an extended relational algebra. The proof of their equivalence is deferred until Chapter 6 so that we can introduce some extended algebra operators which will simplify the proof development. In Chapter 5, we introduce a class of \neg 1NF relations, those that are in "partitioned normal form," and present some extended algebra operators for the \neg 1NF relational model. These operators are closed under the the above class and have additional semantic motivation. Chapter 7 presents our extensions for null values and empty nested relations. We discuss previous work on null values, extend the \neg 1NF model to include null values and empty nested relations, and provide new definitions for the extended algebra operators of Chapter 5 in light of the extension for nulls. In Chapter 8 we define a user oriented language, SQL/NF, for \neg 1NF

relations. Utilizing good language design principles we formulate a high-level database query and manipulation language based on the successful SQL data language for 1NF databases. In Chapter 9 we present our algorithm for achieving Nested Normal Form, incorporating embedded multivalued dependencies and functional dependencies in the design of \rightarrow 1NF relations. Finally, Chapter 10 presents a summary of our contributions and suggestions for future work in this area.

Chapter 2

The 1NF Relational Model

In this chapter, we briefly present the basic characteristics of the 1NF relational model [Cod1, Mai2, Ull]. Portions of this particular condensation are due to Thomas [Tho] and Van Gucht [Van]. We present some basic definitions, the relational calculus and relational algebra, data dependencies and the various normal forms, and some brief comments on null values.

2.1 Basic Definitions

A *domain* is a set of values. If all the values in a domain are atomic (not decomposable) it is a *simple domain*, otherwise it is a *set-valued* or *complex domain*. Given a collection of atomic domains D_1, D_2, \dots, D_n (not necessarily distinct), R is a (1NF) *relation* on these n sets if it is a set of ordered n -tuples $\langle d_1, d_2, \dots, d_n \rangle$ such that $d_i \in D_i, 1 \leq i \leq n$. The value of n is the *arity* of R . A tuple $\langle d_1, d_2, \dots, d_n \rangle$ has n *components*; the i th component is d_i . The domain of an attribute is denoted $\text{DOM}(A)$.

A relation can be viewed as a table with each row corresponding to a tuple and each column representing one component. Columns are usually assigned names called *attribute names*. Individual attributes names are often represented by letters near the beginning of the alphabet, e.g., A, B, \dots , and sets of attributes are represented by letters near the end of the alphabet, e.g.,

..., X, Y, Z . Instead of writing $\{A, B\}$ for the set containing attribute names A and B we use the concatenation AB . Similarly, XY is used to mean $X \cup Y$. Lower case letters near the end of the alphabet are used to represent tuples, e.g. s, t, \dots . Lower case letters p, q, r are used to represent relations and upper case letters P, Q, R are used to represent relation schemes (defined below). When actual names are used, the first letter is capitalized if the name refers to a relation or a set-valued attribute, or left uncapitalized if the name refers to an atomic attribute. Many times, set-valued attributes are represented by following the name with an asterisk, e.g., A^*, B^*, \dots .

We will assume, without loss of generality, that all attributes of our relations are contained in a finite universe of attributes, U . A *relation structure* \mathcal{R} consists of a *relation scheme* R and a *relation* r defined on R , and is denoted $\langle R, r \rangle$. A relation scheme is defined by a *rule* $R = (A_1, A_2, \dots, A_n)$ where $A_i \in U, 1 \leq i \leq n$. The set of attributes in a relation scheme rule R is denoted E_R . For $A \in E_R$, an A -value is an assignment of a value from $\text{DOM}(A)$ to attribute A . Generalizing this notion, an X -value, where $X \subseteq E_R$, is an assignment of values to the attributes in X from their respective domains. Thus, a relation r on scheme R can also be defined as a set of E_R -values.

The projection of relation r onto attributes X is denoted $r[X]$, and similarly, the projection of tuple $t \in r$ onto attributes X is denoted $t[X]$. We also use $t[X]$ to denote an X -value of t when we are talking about an arbitrary

assignment from the respective domains of each attribute in X .

2.2 Relational Calculus

We define a *tuple relational calculus* (TRC). This will form the basis of our extended relational calculus in Chapter 4. We first present a calculus that permits infinite relations and then present “safety” criteria which assures only finite relations can be produced from the calculus formulas.

Formulas in relational calculus are of the form $\{t|\psi(t)\}$, where t is a *tuple variable* denoting a tuple of some fixed length, and ψ is a formula built from atoms and a collection of operators to be defined shortly. We use $t^{(i)}$ to denote the fact that t is of arity i .

The *atoms* of formula ψ are of three types.

1. $s \in r$, where r is a relation name and s is a tuple variable. This atom stands for the assertion that s is a tuple in relation r .
2. $s[i] \theta u[j]$, where s and u are tuple variables and θ is an arithmetic comparison operator ($>$, $=$). This atom stands for the assertion that the i th component of s stands in relation θ to the j th component of u .
3. $s[i] \theta a$ and $a \theta s[i]$, where θ and s are as in (2) above, and a is a constant. The first of these atoms asserts that the i th component of

s stands in relation θ to the constant a , and the second has analogous meaning.

To define the operators of the relational calculus, we need the concept of "free" and "bound" variables from the predicate calculus. An occurrence of a variable in a formula is "bound" if that variable has been introduced by a "for all" or "there exists" quantifier, and the variable is "free" otherwise.

Formulas, and *free* and *bound* occurrences of tuple variables in these formulas, are defined recursively, as follows.

1. Every atom is a formula. All occurrences of tuple variables mentioned in the atom are free in this formula.
2. If ψ_1 and ψ_2 are formulas, then $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, and $\neg\psi_1$ are formulas. Occurrences of tuple variables are free or bound in these formulas as they are free or bound in ψ_1 or ψ_2 , depending on where they occur.
3. If ψ is a formula, then $(\exists s)(\psi)$ and $(\forall s)(\psi)$ are a formulas. Occurrences of s that are free in ψ are bound to $(\exists s)$ in $(\exists s)(\psi)$ and $(\forall s)$ in $(\forall s)(\psi)$. Other occurrences of tuple variables in ψ are free or bound in $(\exists s)(\psi)$ and $(\forall s)(\psi)$ as they were in ψ .
4. Parenthesis may be placed around formulas as needed. We assume the order of precedence is: arithmetic comparison operators highest, then the quantifiers \exists and \forall , then \neg , \wedge , and \vee , in that order.

5. Nothing else is a formula.

A *tuple relational calculus expression* is an expression of the form $\{t|\psi(t)\}$, where t is the only free tuple variable in ψ .

As it stands, this definition of the TRC allows us to define some infinite relations such as $\{t|\neg(t \in r)\}$, which denotes all possible tuples that are not in r , but are of the length we associate with t . As it is impossible to calculate the answer to this query, we must rule out such meaningless expressions. We will do this by restricting consideration to those expression, called "safe," for which it can be demonstrated that each component of any t that satisfies ψ must be a member of $DOM(\psi)$, which is defined to be the set of symbols that either appear explicitly in ψ or are components of some tuple in some relation r mentioned in ψ . This choice of $DOM(\psi)$ is not necessarily the smallest set of symbols we could use, but it will suffice for the 1NF relational model.

We say a tuple calculus expression $\{t|\psi(t)\}$ is *safe* if

1. Whenever t satisfies ψ , each component of t is a member of $DOM(\psi)$.
2. For each subexpression of ψ of the form $(\exists u)(\omega(u))$, if ω is satisfied by u , then each component of u is member of $DOM(\omega)$.
3. For each subexpression of ψ of the form $(\forall u)(\omega(u))$, if any component of u is not in $DOM(\omega)$, then u satisfies ω .

2.3 Relational Algebra

Relational algebra refers to a group of high level operators which are used to manipulate relations. Each of these operators takes one or two relations as input and results in a single relation. The formal definition of the algebra can be found in [Cod3, Ull]. Here we provide only the definitions of the operators themselves.

Since relations are sets of tuples, the usual set operators, union, set difference and Cartesian product apply. These three, along with the special relational operators projection and selection, form a relationally complete set. Relationally complete means that any derivable relations can be retrieved from the database using only this set of operators. We provide also definitions of intersection, theta-join, and natural join which are derivable from the basic operator set. In the following, let r and q be relations.

1. Union—The *union* of r and q , denoted $r \cup q$, is the set of all tuples belonging to either r or s , or both. Relations r and q must be of the same arity, say n , and the j th attribute of each relation must be drawn from the same domain ($1 \leq j \leq n$). When these conditions hold for any two arbitrary relations they are said to be *union-compatible*.
2. Intersection—The *intersection* of r and q , denoted $r \cap q$, is the set of all tuples belonging to both r and q . Relations r and q must be

union-compatible.

3. Difference—The *difference* of r and q , denoted $r - q$, is the set of all tuples in r but not in q . Relations r and q must be union-compatible.
4. Cartesian product—The *Cartesian product* of r and q , denoted $r \times q$, is the set of all tuples that are a concatenation of a tuple from r and a tuple from q .
5. Projection—The *projection* of r over attributes A_1, A_2, \dots, A_n , denoted $\pi_{A_1, A_2, \dots, A_n}(r)$, is the relation obtained by deleting all columns in r except those that are identified by attributes A_1, A_2, \dots, A_n and then eliminating duplicate tuples. In formal proofs we will use also attribute numbers, $1, 2, \dots, k$, where k is the arity of r , instead of attribute names. Attribute number i corresponds to attribute name A_i .
6. Selection—The *selection* of those tuples in r satisfying predicate F , denoted $\sigma_F(r)$, constructs a subset of the tuples in r satisfying F . The predicate F is a formula built from operands that are constants or attribute names (or numbers), arithmetic comparison operators, and the logical operators \wedge , \vee , and \neg .
7. Theta-join—Let A be an attribute in r and B and attribute in q . The

theta-join of r and q , denoted

$$r \bowtie_{A \theta B} q,$$

is the concatenation of a tuple t_r from r and a tuple t_q from q such that $t_r[A]$ has relation θ to $t_q[B]$. When θ is equality the operation is called an equijoin.

8. Natural join—Let r be a relation on scheme R and s a relation on scheme S . Let $X = E_R \cap E_S$ and $Y = E_R \cup E_S$. The *natural join* of r and q , denoted $r \bowtie q$, is the projection onto Y of an equijoin where the equality test is performed on each attribute in X .

The following relationships show how intersection, theta-join, and natural join can be derived from the basic set of operators.

1. $r \cap q = r - (r - q)$.
2. $r \bowtie_{A \theta B} q = \sigma_{A \theta B}(r \times q)$.
3. $r \bowtie q = \pi_Y(\sigma_{r.A_1=q.A_1 \wedge r.A_2=q.A_2 \wedge \dots \wedge r.A_n=q.A_n}(r \times q))$, where A_1, A_2, \dots, A_n are the common attributes of r and q , renamed to be unique by prepending $r.$ or $q.$, as appropriate, and Y is the union of the set of attributes of r and q .

We adopt the following convention for attribute names in cartesian products of relations: We shall use the notation *relation-name.attribute-name*

only when necessary to avoid ambiguity. When no ambiguity results, we shall drop the *relation-name* prefix.

2.4 Data Dependencies

Each relation in a relational database may be expected to reflect certain associations among the stored data. For example, in a relation containing data about employees we might expect each employee number to have associated with it a unique name, address, and telephone number. On the other hand, many employees may have the same name. Such constraints on the contents of a database are termed *data dependencies*.

A relationship in which a single value of one set of attributes is related to the value of a second set of attributes is called a *functional dependency* (FD). Let r be a relation on scheme R , with X and Y subsets of E_R . Relation r satisfies the *functional dependency* $X \rightarrow Y$ if for every pair of tuples t_1 and t_2 , in r , if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

Functional dependencies provide us with a way to define formally the notion of a "key". Let r be any relation on scheme R , and $X \subseteq E_R$. X is a *key* of R if $X \rightarrow E_R$ holds in r , and there does not exist a proper subset Y of X , such that $Y \rightarrow E_R$ holds in r . A *superkey* is any set of attributes which contains a key.

A relationship in which a set of values associated with one set of

attributes is related to the value of a second set of attributes, independent of the other attributes in the relation, is called a *multivalued dependency* (MVD). Let r be any relation on scheme R , with X and Y subsets of E_R and $Z = E_R - XY$. Relation r satisfies the *multivalued dependency* $X \twoheadrightarrow Y$ if for every pair of tuples t_1 and t_2 , in r , if $t_1[X] = t_2[X]$, then there exists a tuple t_3 in r with $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$, and $t_3[Z] = t_2[Z]$.

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

We use several facts about MVDs. Let U be the universe of attributes, X a set of attributes, and M a set of multivalued dependencies. A *dependency basis* for X , denoted $DEP_M(X)$, or $DEP(X)$ when M is understood, is a partition of $U - X$ into sets of attributes Y_1, Y_2, \dots, Y_n , such that if $Z \subseteq U - X$, then $X \twoheadrightarrow Z$ if and only if Z is the union of some of the Y_i 's. For a set M of MVDs, M^+ denotes the *closure* of M , i.e., the set of all MVDs that are implied by M . Given two sets of M and N of MVDs, M is a *cover* of N if $M^+ = N^+$.

Many times we want to work with a *minimum cover* for a set of MVDs.

Definition 2.1: [OY1] Given a set M of MVDs over U , an MVD $X \twoheadrightarrow W$ in M^+ is said to be

- (a) *trivial* if $XW = U$, $W = \emptyset$ or $W \subseteq X$,
- (b) *left-reducible* if $\exists X', X' \subset X$, such that $X' \twoheadrightarrow W$ is in M^+ ,
- (c) *right-reducible* if $\exists W', W' \subset W$, such that $X \twoheadrightarrow W'$ is in M^+ ,
- (d) *transferable* if $\exists X', X' \subset X$, such that $X' \twoheadrightarrow W(X - X')$ is in M^+ .

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

We use $LHS(M)$ to denote the set of left hand sides of the MVDs in a set M of MVDs. Let M^- be the set of all reduced MVDs implied by M , N be a set of MVDs, and M be a minimal cover of N . Then elements in $LHS(M^-)$ are called *keys* of N , and the elements in $LHS(M)$ are called *essential keys* of N . Elements in $LHS(M^-) - LHS(M)$ are called *nonessential keys* of N .

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

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

If a join dependency is equivalent to a set of multivalued dependencies then the \mathcal{R} is an *acyclic* set of schemes. Acyclic schemes have several good properties described in [BFMY, Sac], including the fact that the set of multivalued dependencies which are equivalent to a join dependency are *conflict free*. Properties of *conflict free* dependencies are described in [AC, BeK1, BeK2, Sci1, Sci3, Sci4]. Sciore [Sci3] states that in “real world” situations, every “natural” set of MVDs must be conflict free. Conflict free sets have the desirable property that they allow a unique fourth normal form dependency preserving database scheme; moreover, non-conflict free sets have no such normalization. (Normalization and normal forms are discussed in the next section.) Fixing a particular scheme R , the set of all relations on R that satisfies a set of dependencies D is denoted $SAT_R(D)$, or when R is understood, $SAT(D)$.

Various other dependencies have been proposed and studied, however the FD, MVD, EMVD and JD are the ones most important in the area of database design and normalization. Functional dependencies have been studied in [Hon, Men1], and multivalued dependencies have been studied in [Bis3, Bis4, Fag2, Han, Men2, OY2, Sci3]. Embedded multivalued dependencies have received particular attention by [PP, TKY] and the related *template dependencies* by [Sad, SU1, SU2]. A comprehensive look at join dependencies was done by [ABU].

It is well known that these dependencies satisfy a number of infer-

ence rules. We reproduce the list of [PP], which was assembled from various other sources, with a correction for FD-MVD2. Below it is understood that T, V, W, X, Y, Z represent sets of attributes, and U is the universe of all attributes.

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

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

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

FD4 (Pseudo-transitivity): If $X \rightarrow Y$ and $YV \rightarrow Z$, then $XV \rightarrow Z$.

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

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

MVD0 (Complementation): Given $U = XYZ$ and $Y \cap Z \subseteq X$,
 $X \twoheadrightarrow Y$ iff $X \twoheadrightarrow Z$.

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

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

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

MVD4 (Pseudo-transitivity): If $X \twoheadrightarrow Y$ and $YV \twoheadrightarrow Z$,
then $XV \twoheadrightarrow Z - YV$.

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

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

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

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

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

EMVD0 (Complementation): If $X \twoheadrightarrow Y|Z$, then $X \twoheadrightarrow Z|Y$.

EMVD1 (Projection): If $X \twoheadrightarrow YZ|V$, then $X \twoheadrightarrow Y|V$.

EMVD2 (Root Weighting): If $X \twoheadrightarrow YZ|V$, then $XY \twoheadrightarrow Z|V$.

EMVD3 (Decomposition): If $X \twoheadrightarrow Y|ZV$ and $XY \twoheadrightarrow Z|V$,
then $X \twoheadrightarrow Z|V$.

EMVD4 (Intersection): If $X \twoheadrightarrow Y|Z$ and $X \twoheadrightarrow V|W$ where $Y \cap V \neq \emptyset$
and $Y \cap W \neq \emptyset$, then $X \twoheadrightarrow Y \cap V|Y \cap W$.

EMVD5 (Pseudo-transitivity): If $X \twoheadrightarrow Y|ZVW$ and $YZ \twoheadrightarrow V|XT$ with
 X, Y, Z, V, W disjoint and X, Y, Z, V, T
disjoint, then $XZ \twoheadrightarrow V|YT$.

MVD-EMVD1 (Joinability): $XY \twoheadrightarrow Z$ and $X \twoheadrightarrow Y|Z$ iff $X \twoheadrightarrow Z$.

MVD-EMVD2 (Union): If $XY \twoheadrightarrow VW$ and $XZ \twoheadrightarrow VT$ where $T \subseteq Y$
and $W \subseteq Z$, and $X \twoheadrightarrow Y|Z$, then $X \twoheadrightarrow VWT$.

FD-EMVD1: If $X \twoheadrightarrow Y|Z$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

FD-EMVD2: If $X \twoheadrightarrow Y|Z$ and $XY \rightarrow V$, then $X \twoheadrightarrow YV|Z$.

We note that the inference rules FD1-FD6, MVD0-MVD6, FD-MVD1, and FD-MVD2 form a sound and complete axiomatization of FDs and MVDs [BFH]. Parker and Parsaye-Ghomi [PP] showed that there can be no finite set of inference rules for EMVDs, based on the assumption that an arbitrary number of attributes is available. Should the number of attributes be fixed, there must be a complete set of rules, although the cardinality of this set will be large (and as of yet, still undiscovered).

2.5 Normal Forms

A "normal form" is a restriction on the database scheme that precludes certain undesirable properties from the database. Most of these undesirable properties deal with update (including insert and delete) anomalies and redundancy in the database. A number of different normal forms for relation schemes with dependencies have been defined so that some of the anomalies and redundancy are no longer present in the database. Since they will play a part in 1NF database normalization, we will consider two normal forms here: Boyce-Codd Normal Form [Cod3, LeP] and Fourth Normal Form [Fag2]. The two definitions that follow are slightly different than usual as they do not require that the

relation be in 1NF; that is, the all domains are simple. As Kobayashi [Kob] pointed out, the 1NF restriction is strictly not needed in the definition of these normal forms, however traditional dependency theory is not capable of dealing with complex domains and so the 1NF restriction is added to the definitions.

A relation scheme R with FDs F is said to be in *Boyce-Codd Normal Form* (BCNF) with respect to F , if whenever $X \rightarrow A$ holds in any relation r on scheme R , and $A \notin X$, then $X \rightarrow E_R$ holds in r ; that is, X is a key.

Now, let R be a relation scheme and D a set of FDs and MVDs. We say that R is in *Fourth Normal Form* (4NF) with respect to D , if whenever there is a MVD $X \twoheadrightarrow Y$, where $Y \neq \emptyset$, $Y \not\subseteq X$, and $XY \neq E_R$, then $X \rightarrow E_R$; that is, X is a key. We note that 4NF implies BCNF.

Other normal forms include 2NF and 3NF which are defined based on schemes being free from partial and transitive dependencies, respectively [Cod1], an improved 3NF which removes superfluous attributes from schemes [LTK], *Project/Join Normal Form* (PJ/NF) based on the two operators projection and natural join [Fag3], and *Domain/Key Normal Form* (DK/NF), an ultimate normal form based only on domain and key constraints (as yet unattainable in general) [Fag1]. Note that 2NF and the two 3NFs are defined when only FDs are present, PJ/NF for JDs, MVDs and FDs, and DK/NF for arbitrary constraints.

The goal of *database design* is to produce a set of schemes which ex-

hibits the good properties espoused by the various normal forms [LST, ZM]. Two approaches are generally used in the design algorithms: decomposition and synthesis. The *decomposition* method assumes a *universal relation* [FMU] containing all attributes of the database exists, and then proceeds to decompose this scheme based on the dependencies to be satisfied, and the normal form to be achieved. A decomposition of a scheme is its replacement by a collection $\rho = \{R_1, R_2, \dots, R_n\}$, where $E_{R_i} \subset E_R$, $1 \leq i \leq n$, and $\cup_{i=1}^n E_{R_i} = E_R$. The decomposition ρ is a *lossless join decomposition* with respect to a set of dependencies D if for every relation r on scheme R satisfying D :

$$r = r[E_{R_1}] \bowtie r[E_{R_2}] \bowtie \dots \bowtie r[E_{R_n}].$$

The *synthesis* method starts with the attributes in each dependency and synthesizes a set of schemes which meet the goals of the normal form. In the next chapter, we will see that there are disadvantages to vertical decomposition and that $\neg 1NF$ is a viable alternative.

2.6 Null Values

Throughout this discussion we have ignored the concept of null values in the database. *Null values* indicate the lack or nonexistence of information in the database. They do not lend themselves well to the rigorous analysis that applies to most other aspects of the relational model. However, there has been substantial research in this area which is summarized in Chapter 7. It is in that chapter that we discuss the role of the null value in both the 1NF and $\neg 1NF$

relational models. Until then we assume that null values are not allowed in the database.

Chapter 3

The \neg 1NF Relational Model

In this chapter, we describe several aspects of the \neg 1NF relational model. We examine various database models which have been proposed for dealing with non-atomic domains and define the particular model we will be using in this dissertation. We will then take a brief look at previous work done in the areas of query languages, dependency theory, normal forms, and applications for \neg 1NF relations.

3.1 \neg 1NF Database Models

One of the chief benefits derived from working with the relational approach to databases is that it can be couched within the formalism of first-order predicate logic. As a result many important issues can be addressed mathematically when one assumes the database is relational. However, when the 1NF assumption is not made, we need an analogous formalism that will serve the \neg 1NF approach. There are several existing models which have the characteristics we require.

The *database abstractions* of Smith and Smith [SmS] model aggregation and generalization of data. We are interested in the ability to aggregate simple domains into complex domains, but not the classification of domains that generalization allows. Our extensions are developed for a simpler model in which generalization is not allowed. However, generalization can be simu-

lated in our model by using a "class" attribute to distinguish tuples in different generalization categories. For example, the "vehicle" domain is a generalization of "car," "truck," and "bus" domains. We would add an attribute "vehicle-type," to distinguish data elements of these different classes within the "vehicle" domain. Abiteboul [Abi] extends this work by introducing *disaggregation*, an inverse of the aggregation concept. Disaggregation can be regarded as a columnwise index which maps each value in a particular column into a tuple. The concept of indexing is taken to the extreme in Orman's *indexed data sets* [Orm]. In this model the values of one attribute are used as an index for the values of other attributes using binary associations. This has the effect of partitioning relations via the indices.

A more restricted model for non-first-normal-form relations is the *Verso* model [B+, AB1], where instances are defined over a *format*. A *format* is recursively defined by:

- (i) let X be a finite string of attributes with no repeated attribute, then X is a *format* over the set X of attributes, and
- (ii) let X be a finite string of attributes with no repeated attribute, X non-empty, and f_1, f_2, \dots, f_n some formats over Y_1, Y_2, \dots, Y_n , respectively, such that the sets X, Y_1, Y_2, \dots, Y_n , are pairwise disjoint, then the string $X(f_1)^*(f_2)^* \cdots (f_n)^*$ is a format over the set $XY_1Y_2 \cdots Y_n$.

Let $tup(X)$ be a set of X -values. An instance over a format f , denoted $inst(f)$,

is recursively defined by:

- (i) if $f \equiv X$ then $inst(f)$ is a finite subset of $tup(X)$, and
- (ii) if $f \equiv X(f_1)^*(f_2)^*\dots(f_n)^*$ then I is in $inst(f)$ if and only if
 - (a) I is a finite subset of $tup(X) \times inst(f_1) \times inst(f_2) \times \dots \times inst(f_n)$, and
 - (b) if $\langle u, I_1, I_2, \dots, I_n \rangle$ and $\langle u', I'_1, I'_2, \dots, I'_n \rangle$ are in I then $u \neq u'$ or $\langle u, I_1, I_2, \dots, I_n \rangle = \langle u', I'_1, I'_2, \dots, I'_n \rangle$.

The (a) condition states that I is atomic on the attributes in X and not atomic on the "attributes" f_1, f_2, \dots, f_n . The (b) condition forces X to be a key. This is a large restriction on what 1NF relations can be. We will look at the advantages of such a restriction in section 3.4.

The *format model* of Hull and Yap [HY] recursively builds *formats* using the three data constructs: *collection*, *composition*, and *classification*. Composition and classification are closely related to aggregation and generalization, respectively, of [SmS]. Collection allows one to specify the formation of sets of objects, all of a given type. The *database logic* of Jacobs [Jac1-3] is a framework for a heterogeneous database which can serve the relational, hierarchical, and network approaches. Kuper and Vardi [KV1] have a modified database logic which models also *virtual records*, introducing cyclicity into the schema level and solves the problems of noncomputable queries present in Jacob's logic.

Kuper and Vardi's logic specifies database instances by *r-values* for the data space, and *l-values* for the address space. An instance of the database is a set of *l-values* and their associated *r-values*. This makes query languages very cumbersome to use since the user must know about and manipulate "conceptual addresses" throughout the database. Additionally, database logic is too powerful for our purposes. In particular, Kuper and Vardi [KV2] show that their algebra is equivalent to an algebra which includes the power set operator, which is not expressible (see Appendix A) with the basic relational operators or the extended algebra operators for \neg 1NF relations (see Chapter 4). Other more powerful models include the *Graph Data Model* described in [Kun] and the various semantic data models such as the *Functional Data Model* described in [Shi].

We follow the lead of Fischer and Thomas [FT] and adopt a formalism adapted from the database logic of Jacobs. Some of the following description of our \neg 1NF model is taken from [FT].

A *database scheme S* is a collection of rules of the form

$$R_j = (R_{j_1}, R_{j_2}, \dots, R_{j_n}).$$

The objects $R_j, R_{j_i}, 1 \leq i \leq n$, are attributes. R_j is a *higher order attribute* if it appears on the left hand side of some rule; otherwise it is *zero order*. Each zero order attribute has an associated *domain* from which the attributes values are drawn. The attributes on the right hand side of rule R_j form a set denoted

Employee

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

Figure 3-1. A sample relation on the Emp scheme.

E_{R_j} , the elements of R_j . As with any set, attributes on the right hand side of the same rule are unique, and to avoid ambiguity, no two rules can have the same attribute on the left hand side.

To illustrate this, consider the following database scheme.

Emp = (ename, Children, Skills),
 Children = (name, dob),
 Skills = (type, Exams),
 Exams = (year, city).

In this scheme each employee has a set of children each with a name and birthdate, and a set of skills, each with a skill type and a set of exam years and cities, when and where the employee retested his proficiency at the skill. A sample relation is shown in the relation in Figure 3-1.

In this example, the higher order attributes are *Emp*, *Children*, *Skills* and *Exams*. All others are zero order attributes. An attribute R_j is *external* if

it appears *only* on the left hand side of some rule, otherwise it is *internal*. Thus in the above example, Emp is external while all other attributes are internal.

We often are concerned with an individual table or relation scheme, not with the entire database. Let R_j be an external attribute in database scheme S . The rules in S which are *accessible* from R_j form a subscheme of S , defined as follows:

1. $R_j = (R_{j_1}, R_{j_2}, \dots, R_{j_n})$ is in the subscheme, and
2. When a higher order attribute R_k is on the right hand side of some rule in the subscheme, the rule $R_k = (R_{k_1}, R_{k_2}, \dots, R_{k_n})$ is also in the subscheme.

A subscheme is called a *relation scheme* if in addition:

3. No zero order attribute appears on the right hand side of two different rules in the scheme.

For example, consider the employee database scheme. The subscheme starting with Emp contains the rules for Emp, Children, Skills and Exams, and the subscheme starting with Children contains only the rule for Children. Since there are no zero order attributes appearing in more than one rule, both of these subschemes are also relation schemes.

A 1NF database scheme is a collection of rules of the form $R_j =$

$(R_{j_1}, R_{j_2}, \dots, R_{j_n})$ where all the R_{j_i} are zero order. A \neg 1NF scheme, however, may contain any combination of zero or higher order attributes on the right hand side of the rules as long as the scheme remains nonrecursive[†]. Note that a nested relation is represented simply as a higher order attribute on the right hand side of a rule.

As in the 1NF relational model, let R be an attribute appearing in a database scheme S . An *instance* of R , written r , is an ordered pair of the form $\langle R, V_R \rangle$ where V_R is a value for attribute R . When R is a zero order attribute, V_R is just any value from the domain of R . When R is a higher order attribute, V_R must be expanded in terms of the attributes on the right hand side of rule R . We will omit the attribute name in an instance specification when the name is understood from the context.

Two schemes R_i and R_j are equal if they are comprised of the same rules. In order for two structures to be equal, their schemes and instances must be equal. Two instances r_1 and r_2 of equal relation schemes R_1 and R_2 are equal if the identity mapping is an isomorphism from r_1 to r_2 .

3.2 Formal Query Languages for \neg 1NF Relations

Extensions to relational calculus and relational algebra languages to support \neg 1NF relations began by adding one-level nest and unnest operators to the

[†] Recursive schemes are beyond the scope of this dissertation.

basic relational algebra. Jaeschke and Schek [Jae1, JS] defined one-level nest and unnest operators and extended selection predicates to include containment and subset comparison operators. Özsoyoğlu, Özsoyoğlu, and Matos [OOM2] extend the relational algebra and calculus for set-valued attributes (single-attribute, one-level nests) and aggregate functions (e.g., MAX, SUM, AVG). Language extensions for aggregate functions can also be found in [Eps, Klu]. Arisawa, Moriya, and Miura [AMM] take single-attribute, one-level nesting to the extreme by nesting every attribute of the relation and studying operations on these relations.

Multi-attribute, multi-level nesting was first studied by Fischer and Thomas [FT, TF] and Abiteboul and Bidoit [AB2]. Fischer and Thomas extend the basic relational algebra with multi-attribute nest and unnest operators and study the interaction of these operators with the traditional algebra operators. Due to the Verso model's more restricted nature (see section 3.1), Abiteboul and Bidoit introduce extended algebra operators in addition to nest and unnest, which maintain the underlying semantics of their relations. Some of these operators are refined and formally presented in Chapter 5. An attempt is also made in [AB2] to define a select operator which operates in a recursive manner to select tuples from nested relations. Jaeschke [Jae2] also has a proposal for an algebra similar to [FT], but with additional *local* algebra operators which operate within nested relations that occur in every tuple. The full power of a recursive algebra in which operators can be nested within other algebra

operators has been proposed in [Jae3, Sch2, ScS1, ScS2].

The algebra operators of the recursive algebras and extensions to include set comparison operators can all be expressed in terms of a basic relational algebra and the addition of multi-attribute nest and unnest. This is the algebra we present in Chapter 4, along with a new calculus of equivalent power.

3.3 Dependencies for \neg 1NF Relations

Two primary directions have been taken in the area of dependency theory as applied to \neg 1NF relations. One direction has been to define new dependencies directly on \neg 1NF relations, while the other direction involves using dependencies defined on 1NF relations and investigating their consequences in the nested counterparts of those relations.

3.3.1 New Dependencies for \neg 1NF Relations

Some researchers [Kob, Mak, Tho] have extended the usual definitions of dependency by simply extending the notion of equality expressed in these definitions to include set-equality when higher order attributes are involved. For example, in Figure 3-1, the extended FDs, $\text{ename} \rightarrow \text{Children}$ and $\text{ename} \rightarrow \text{Skills}$, hold in the Employee relation. Furthermore, we would expect them to hold on any relation over the Emp scheme. These extended dependencies are generally unable to cope with nested relations. Looking at Figure 3-1 again, we see that the extended FD, $\text{type} \rightarrow \text{Exams}$, holds in each nested Skills relation. However, if

we unnested Employee on the Skills attribute, that same FD would no longer hold. Thus, the concept of "local" dependency [Tho, Van] was introduced. A dependency is *local* if it holds within a nested relation. If the dependency holds in the nested relation of each tuple, throughout the relation it is nested in, then the dependency is said to be *uniformly local*. The usual dependency which must hold on an entire relation is now called *global*. Several interesting results were discovered by Thomas [Tho] concerning the interaction of global and uniformly local dependencies with the nest operator.

A new dependency, directly involving the higher order attributes of a relation, was introduced by Van Gucht and Fischer. They define the *strong functional dependency* (SFD) for one-level schemes [FV1] and the *generalized functional dependency* (GFD)[†] for multi-level schemes [VF]. Since GFDs include SFDs as a subclass, we will describe the GFD only. Let S be a scheme, $H(S)$ the higher order attributes of S , and $A(S)$ the lower order attributes of S . First, we need a recursive definition of intersection for multi-level schemes called *overlap*. Let v_1, v_2 be tuples of s on relation scheme S and let $Y \in H(S)$. We say that v_1 and v_2 *overlap* on Y , denoted $v_1(Y) \text{ ovp } v_2(Y)$ if and only if

(1) $v_1(Y) \cap v_2(Y) \neq \emptyset$, or

(2) there exist tuples $t_1 \in v_1(Y)$ and $t_2 \in v_2(Y)$ such that $t_1[A(Y)] = t_2[A(Y)]$ and $t_1(M) \text{ ovp } t_2(M)$ for all $M \in H(Y)$.

[†] This GFD is different than one used in [Ull, SU1] for generalizing FDs for 1NF databases.

Let s be a relation on scheme S , $V, Z \subseteq E_S$, $W \subseteq H(S)$. We say that s satisfies the *generalized functional dependency* $V \langle W \rangle \rightarrow Z$ if and only if for any two tuples $t_1, t_2 \in s$ such that $t_1(V) = t_2(V)$ and $t_1(M) \text{ ovp } t_2(M)$ for all $M \in W$, we have $t_1(Z) = t_2(Z)$.

When $W = \emptyset$, a GFD is nothing but an ordinary FD. GFDs are used in [VF] to characterize a class of $\neg 1NF$ relations called "permutable nested relations," and in [Van] to characterize the semantics of some $\neg 1NF$ relations.

3.3.2 Using Dependencies on 1NF Relations

Several proposals have been made for using dependencies defined on 1NF relations for the purpose of structuring $\neg 1NF$ relations. Özsoyoğlu and Yuan [OY1] use functional and multivalued dependencies to determine how to set up a "good" set of $\neg 1NF$ relations, which takes advantage of the given dependencies. For example, let U be a set of attributes, X , Y , and Z a partition of U , and r a 1NF relation on scheme $R = (U)$. If the multivalued dependency $X \twoheadrightarrow Y | Z$ holds in r then consider the relation s with the Z attributes forming one nested relation and the Y attributes forming another nested relation for each X value. Relation s is a $\neg 1NF$ relation with several good properties. First, X is a key for s , giving a unique tuple in s for each X -value. Second, the Y and Z nested relations are independently updatable; adding a value to Z (Y) automatically enforces the underlying MVD by matching all values in Y (Z) with the new value added. Third, we can nest the underlying relation in any order,

first by Z then by Y , or in the reverse order, and achieve the same relation s . We will discuss these issues further in section 3.4, where they play a part in normal forms for $\neg 1NF$ relations. MVDs (and EMVDs) can be expressed also as *first-order hierarchical dependencies* and *generalized hierarchical dependencies* [Del]. These dependencies more easily show the hierarchical structure of a set of MVDs, but do not provide any more power in the $\neg 1NF$ design process.

Kambayashi, et al. [KTT, KTTY], give procedures for designing nested relations using a set of constraints consisting of one join dependency, functional dependencies satisfied by each component of the join dependency, and a hierarchy of related attribute sets. A join dependency can be used in a similar way that we used the MVD above to achieve a nested scheme. Functional dependencies are used to do further nesting within each nested relation of the scheme produced using the JD. Note that if the FD, $X \rightarrow Y$ holds in a relation and we nest on Y , then each nested relation will be a singleton set. This is clearly a wasted operation, and so [KTT] proposes a scheme where a chain of FDs (using the transitivity property of FDs) is used and the right hand sides of the FDs are successively nested to achieve the most redundancy reduction possible. Related attribute sets are simply collections of attributes that are grouped together so they can be accessed as a single unit. The problem with this approach is that a single JD is not enough to characterize the structure of nested relations to anything more than one-level deep.

A new dependency on 1NF relations was discovered by [JS] to characterize exactly when two nest operations will commute. The dependency is called a *weak multivalued dependency* (WMVD) and is defined as follows. Let U be a set of attributes and let X, Y, Z be subsets of U such that $Z = U - XY$. A WMVD, denoted $X-w \rightarrow Y$, is a template dependency with hypothesis rows t_1, t_2 , and t_3 , and a conclusion row t_4 such that:

1. $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
2. $t_1[Y] = t_2[Y]$
3. $t_1[Z] = t_3[Z]$
4. $t_4[Y] = t_3[Y]$
5. $t_4[Z] = t_2[Z]$

In tableau form, $X-w \rightarrow Y$ is the WMVD

	X	$Y - X$	Z
t_1 :	x	y	z
t_2 :	x	y	z'
t_3 :	x	y'	z
t_4 :	x	y'	z'

A relation r on scheme $R = (U)$ satisfies $X-w \rightarrow Y$ if r satisfies the TD $(t_1, t_2, t_3)/t_4$ given above. In contrast, the ordinary MVD, $X \twoheadrightarrow Y$ would correspond to the TD $(t_2, t_3)/t_4$.

The major contribution of the WMVD is its characterization of when nests commute. Using U, X, Y, Z , and r as above, [JS] showed that $X-w \rightarrow Y$ holds in r if and only if nesting on Y and Z commutes. This of course was

for single-attribute, single-level nesting. Thomas [Tho] extended this result to nesting on arbitrary structures, and Fischer and Van Gucht [FV3] extend Thomas' results to more than two nest operations. [FV3] provides also a sound and complete axiomatization of WMVDs, and [Van] extends this to a sound and complete axiomatization of a mixed system of MVDs and WMVDs.

3.4 Normal Forms for \rightarrow 1NF Relations

3.4.1 Horizontal Decomposition

Researchers have suggested that horizontal decomposition or nesting can be used instead of vertical decomposition to improve database design. Horizontal decomposition[†] was suggested by Furtado [Fur] to improve schemes that are not dependency preserving BCNF. A dependency is preserved by a decomposition if the attributes of the dependency exist in one scheme or the dependency is implied by the non-trivial dependencies whose attributes are subsets of a scheme. An example used in [Fur, Sci1, Ull, Van] to illustrate this is as follows. Consider the relation scheme $R = (\text{city}, \text{st}, \text{zip})$. A tuple $\langle c, s, x \rangle$ is in a relation on scheme R if city c has a building with street address s , and x is the zip code for that address in that city. We have the following FDs:

- $\{\text{city}, \text{st}\} \rightarrow \text{zip}$
- $\text{zip} \rightarrow \text{city}$.

[†] A database model employing horizontal partitioning was developed around the concept of "quotient relations" by Furtado and Kerschberg [FK]. An algebraic specification for quotient relations as an abstract data type is found in [Tom].

The BCNF decomposition of this scheme is

- $R_1 = (\text{st}, \text{zip})$
- $R_2 = (\text{zip}, \text{city})$.

This scheme is not dependency preserving since the attributes of $\{\text{city}, \text{st}\} \rightarrow \text{zip}$ are not included in either R_1 or R_2 and the only FD which is included, $\text{zip} \rightarrow \text{city}$, does not imply $\{\text{city}, \text{st}\} \rightarrow \text{zip}$. Thus, we can have legal instances of relations on the decomposed schemes that do not join to a legal instance of the original scheme.

[Fur] suggests horizontal partitioning of scheme R_1 by city. Then in each block created by the partitioning the dependency, $\text{st} \rightarrow \text{zip}$, holds, and each block is disjoint from all others. Thus, we can assure that the dependency, $\{\text{city}, \text{st}\} \rightarrow \text{zip}$, is enforced by checking that the induced dependency, $\text{st} \rightarrow \text{zip}$, holds in each block, and by checking that zip codes remain partitioned among the blocks.

When we allow nested relations, then even the initial vertical decomposition is not necessary. The scheme $R = (\text{city}, ZS)$, $ZS = (\text{zip}, \text{st})$, would have the same advantages described above, with blocks now corresponding to nested relations, and without the disadvantage of having two relations. However, we can go further. Since, $\text{st} \rightarrow \text{zip}$, holds in each nested relation each value of st is associated with exactly one value of zip . Therefore, we can nest all st values for a particular zip value into a nested relation, obtaining the scheme $R = (\text{city}$,

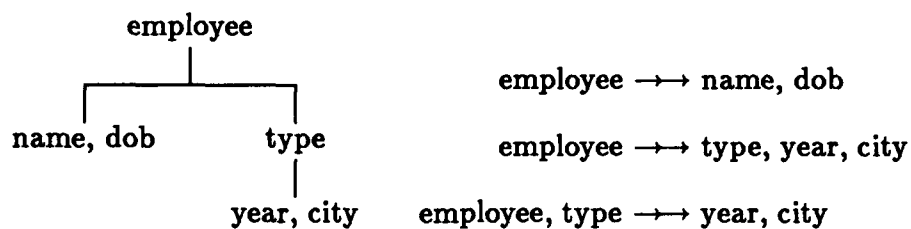


Figure 3-2. Scheme tree and implied MVDs for employee database.

ZS), $ZS=(zip, ST^*)$, $ST^*=(st)$.

3.4.2 Nested Normal Form

Özsoyoğlu and Yuan [OY1] introduced the first comprehensive approach to normalization for \neg 1NF relations. They consider nested relations whose schemes are structured as trees, called *scheme trees*, and introduce a normal form for such relations, called *nested normal form* (NNF). A *scheme tree* is a tree whose vertices are labeled by pairwise disjoint sets of zero order attributes, where the edges of the tree represent MVDs between the attributes in the vertices of the tree. These MVDs allow a 1NF relation to be represented as a \neg 1NF relation with the good properties discussed in section 3.3. The scheme tree and associated MVDs for the Emp scheme are shown in Figure 3-2.

Formally, let U be a set of zero order attributes, T be a scheme tree, and $e = (u, v)$ be an edge of T . Let $A(v)$ be the union of all ancestors of v , including v , $D(v)$ be the union of all descendants of v , including v , and $S(T)$ be the union of all attributes in T . Then the MVD represented by the edge e is

$A(u) \twoheadrightarrow D(v)$ in the context of $S(T)$. Also, let $MVD(T)$ be the set of MVD's represented by the edges of T .

Definition 3.1: [OY1] Let T be a scheme tree, and u_1, u_2, \dots, u_n be all the leaf nodes of T . Then the *path set* of T , denoted $P(T)$, is $\{A(u_1), A(u_2), \dots, A(u_n)\}$. Note that, for a leaf node u , $A(u)$ is the union of all the nodes in the path from the root of T to u in T .

The following proposition gives some properties of a scheme tree.

Proposition 3.1: [OY1] If T is a scheme tree, then

1. $P(T)$ is an acyclic database scheme,
2. $MVD(T) \iff \bowtie (P(T))$, and
3. $MVD(T)$ is conflict free. □

Let T be a scheme tree with respect to M , where $S(T) \subseteq U$, and (u, v) be an edge in T . Assume there is a key X of M [OY2] such that there exists $Z \in DEP(X)$ and $D(v) = Z \cap S(T)$. Then, v is said to be a *partial redundant* in T with respect to X if $X \subset A(u)$. The MVD, $X \twoheadrightarrow D(v)$ in the context of $S(T)$ is a *partial dependency* in $S(T)$. Similarly, if there exists some sibling nodes v_1, v_2, \dots, v_n of v in T such that $W = \bigcup_{i=1}^n D(v_i)$, $X \subset A(u)W$, and M does not imply $XW \twoheadrightarrow D(v)$ in the context of $S(T)$, then v is said to be *transitive redundant* with respect to X in T . In this case, the MVD, $X \twoheadrightarrow D(v)$, in the context of $S(T)$, is said to be a *transitive dependency* in $S(T)$.

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

$$FK(V) = \{V \cap X \mid X \in LHS(M) \text{ and } V \cap X \neq \emptyset,$$

and there is no $Y \in LHS(M)$ such that $X \cap V \supset Y \cap V \neq \emptyset\}$.

Given a set M of MVDs on attributes U , [OY1] gives an algorithm to decompose U into a set of scheme trees which do not have partial or transitive redundancies and does not divide keys in the trees. A normal scheme tree is defined as follows.

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

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

The method proposed in [OY1] uses MVDs and the MVD counterpart of FDs (via rule FD-MVD1) as input to the NNF decomposition algorithm. In [YO], the authors have combined FDs and MVDs into an *envelope* set of dependencies. They propose that this envelope set could be used as input to a slightly modified NNF algorithm which would then take into account the different semantics of FDs and MVDs. Using the algorithm in [OY1], singleton

sets are likely to appear when FDs are used to perform the decomposition. We propose a new method for achieving nested normal form which takes into account the different semantics of FDs in Chapter 9.

3.5 \neg 1NF Applications

In this section we sample a variety of applications for \neg 1NF relations. We will describe and, where space permits, show an example of \neg 1NF relations to model office forms, complex objects and CAD, statistical databases, information retrieval systems, and a relational operating system interface.

3.5.1 Office Forms

Implementing office forms in a database system are discussed in [AH, KTW, SLTC]. In [AH], the *format model* is used as a foundation for studying the structure of forms as they arise in office information systems. Form systems based on \neg 1NF relations are described in [KTW]. They propose a design methodology for conceptual modeling of \neg 1NF relations, especially to represent the semantic concepts needed for form systems, and give an overview of a prototype implementation of a form system at the University of Vienna. A formal means for specification of forms processing is presented in [SLTC]. Figure 3-3 shows how an invoice form would appear as a \neg 1NF relation. Note that this is a user view; the stored data would not include amount and total columns as these are derived from the other data in the relation.

Invoices

cname	caddress	Orders				total
		prod-no	qty	price	amount	
Smith	Chicago	102	10	1.30	13.00	80.00
		210	1	67.00	67.00	
West	Auburn	102	5	1.30	6.50	20.80
		213	43	0.10	4.30	
		456	10	1.00	10.00	

Figure 3-3. An invoice form represented as a 1NF relation.

3.5.2 Complex Objects and CAD

Complex objects and CAD applications are obvious candidates for the 1NF model. Issues involved in using the relational model for these applications are discussed in [BaKh, BaKi, HL, Lor, ML]. Most of this research is involved in how to model complex objects using the traditional relational model. An example from [Lor] will illustrate how we can use 1NF relations to our advantage in this environment. Let us consider the design of electronic components. A particular component is called an entity. An entity can comprise several other entities at a different level. Consider, for example, a 4-AND entity built out of three elementary 2-AND gates. A design for the 4-AND entity is illustrated in Figure 3-4.

The description, both topological and graphical, can be mapped into relations. Figure 3-5 shows the contents of the 1NF relations for a simple design. The relation Entity contains, for each entity, its unique identification

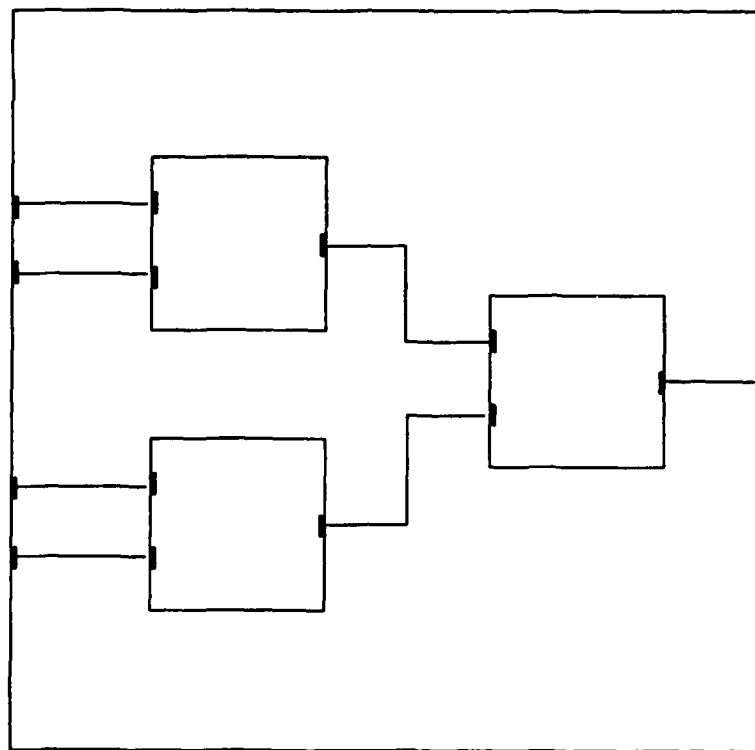


Figure 3-4. Design of a 4-AND component.

number and its name. The relations Geometry and Pins specify, for each entity, its exterior representation. Geometry specifies the lines drawn from (x_1, y_1) to (x_2, y_2) while the relation Pins specifies the exterior pins of the entity: pin number, class (input or output) and position. The internal contents of an entity are specified in terms of other entities that are used to build the higher level entity. An instance of an entity used inside another entity is called a block. The relation Blocks specifies the blocks used inside an entity: the number of each block, the type (the identifier of the entity of which this block is an instance), and some graphical information such as position and scale. The rela-

Entity

id	name
100	2-AND
200	4-AND

Blocks

id	no	type	X	Y	...
200	1	100	10	10	
200	2	100	10	30	
200	3	100	30	20	

Geometry

id	X1	Y1	X2	Y2
100	0	0	0	10
100	0	10	10	10
100	10	10	10	0
100	10	0	0	0
200	0	0	0	50
200	0	50	50	50
200	50	50	50	0
200	50	0	0	0

Pins

id	no	class	X	Y
100	1	IN	0	3
100	2	IN	0	7
100	3	OUT	10	5
200	1	IN	0	13
200	2	IN	0	17
200	3	IN	0	33
200	4	IN	0	37
200	5	OUT	50	25

Conx-segments

id	cno	X	Y
200	6	25	15
200	6	25	23
200	7	25	35
200	7	25	27

Connections

id	cno	block1	pin1	block2	pin2
200	1	0	1	1	1
200	2	0	2	1	2
200	3	0	3	2	1
200	4	0	4	2	2
200	5	0	5	3	3
200	6	1	3	3	1
200	7	2	3	3	2

Figure 3-5. 1NF relations for circuit design.

Entity		Geometry				Pins				Blocks				Connections									
id	name	X1	Y1	X2	Y2	no	class	X	Y	no	type	X	Y	cno	b1	p1	b2	p2	Conx-segs				
																			X	Y			
100	2-AND	0	0	0	10	1	IN	0	3														
		0	10	10	10	2	IN	0	7														
		10	10	10	0	3	OUT	10	5														
		10	0	0	0																		
200	4-AND	0	0	0	50	1	IN	0	13	1	100	10	10	1	0	1	1	1					
		0	50	50	50	2	IN	0	17	2	100	10	30	2	0	2	1	2					
		50	50	50	0	3	IN	0	33	3	100	30	20	3	0	3	2	1					
		50	0	0	0	4	IN	0	37														
						5	OUT	50	25														
																						25	15
																						25	23
																				25	35		
																				25	27		

Figure 3-6. -1NF relation for circuit design.

tion Connections shows the topology of the connections inside an entity. Each row gives the connection number and the two block/pins that are connected. Graphically, a connection is represented as a line built out of one or several segments. A relation Conx-segments contains a row for every intermediate point in a connection between two block/pins; if a connection is made out of a single line segment there is no corresponding row in Conx-segments.

Six relations are needed to represent the circuit design database, even though there is only a single object being modeled. A -1NF design for this database uses only one relation, as in Figure 3-6. Each tuple of this relation contains all of the data on each entity: its id, name, geometry, pins, blocks, and connections. The user can more easily see the entire design of an entity, and queries will be easier to formulate, since only one relation need be queried.

SUM-SALARY-OF EMPLOYEES	DIV:		
	div1		div2
	DEPT: man personnel		DEPT: acct
AGE:	SUM-SAL:		
[18,30]	100K	150K	290K
[31,40]	200K	300K	400K
[41,60]	250K	350K	250K

Figure 3-7. An example summary table: SUM-SALARY-OF-EMPLOYEES

3.5.3 Statistical Databases

Statistical databases are a natural candidate for $\neg 1NF$ relations since grouping of data is accomplished so that statistics can be applied to them. Modeling of statistical database applications was done by [Joh, OO2]. A query language and physical organization techniques for a statistical database are described in [OOM1, OO1, OO3]. In Figure 3-7, we show an example of a “summary table” from [OO3]. This table shows, for each age-group, the sum of the salaries of employees in each department of each division of some company. This same data can be represented as a $\neg 1NF$ relation as shown in Figure 3-8.

3.5.4 Relational Operating System Interface

Korth and Silberschatz [Kor, KS] propose extending the relational model to support an operating system interface. The ability to use a $\neg 1NF$ model greatly enhances this idea. For example, one function of an operating system is to allow

Sum-Salary-of-Employees

AGE	DIV-Salaries		
	DIV	DEPT-Salaries	
		DEPT	SUM-SAL
[18,30]	div1	man	100K
		personnel	150K
	div2	acct	290K
[31,40]	div1	man	200K
		personnel	300K
	div2	acct	400K
[41,60]	div1	man	250K
		personnel	350K
	div2	acct	250K

Figure 3-8. Summary table as a \neg 1NF relation.

users to communicate with each other by exchanging messages. Such a mail system could be represented by the two relations in-mail on scheme (sender, cc-list, subject, date-received, text) and out-mail on scheme (to, cclist, subject, date-sent, text). Mail is read by querying the in-mail relation, and mail is sent by adding a tuple to the out-mail relation. The attributes cc-list and text are set-valued attributes. The cc-list attribute contains all addressees which will get a copy of the message, and text can be broken down into lines or words. If a 1NF view of this relation were needed, each addressee and each line of text would force another tuple to be added to the mail relations. To overcome the additional redundancy this causes, we would have to decompose the relations in the database, causing the user's view of mail to become fragmented and complicating the use of the mail system.

Books

Authors	Title	Price	Descriptors
A1, A2	T1	P1	D1, D2
A2	T1	P2	D1, D2
A1	T1	P1	D1, D2, D3

Figure 3-9. Books table as a \neg 1NF relation.

3.5.5 Information Retrieval Systems

There is a trend towards integrating database management systems and information retrieval systems. [GP, Mac, PS, Sch1, SP] describe methods for enhancing relational database systems to support the information retrieval application. Most of this work is concerned with textual data, however, pictorial and graphical data have some similar support requirements. The *Advanced Information Management* (AIM) project has been running at the IBM Heidelberg Scientific Center since 1978. This project is testing the feasibility of integrating the management of formatted and unformatted data into a \neg 1NF relational database. Figure 3-9 shows a \neg 1NF book inventory table in the style of [SP], while the normalized 1NF version of this table, requiring three relations, is shown in Figure 3-10. There are also many common information retrieval requests which are hard to formulate on the basis of the data structure in Figure 3-10, such as

Display title and price of books described by both descriptors D1 and D2 and written by author A1.

Figure 3-11a gives an SQL-like [C+] formulation of this query. Intu-

Book			Author		Descriptor	
Bno	Title	Price	Bno	Author	Bno	Descriptor
1	T1	P1	1	A1	1	D1
2	T1	P2	1	A2	1	D2
3	T1	P1	2	A2	2	D1
			3	A1	2	D2
					3	D1
					3	D2
					3	D3

Figure 3-10. 1NF relations corresponding to Books table of Figure 3-9.

```

SELECT Title, Price
FROM   Book, Author,
       Descriptor X, Descriptor Y
WHERE  Author = A2
AND    Author.Bno = Book.Bno
AND    Book.Bno = X.Bno
AND    X.Descriptor = D1
AND    Book.Bno = Y.Bno
AND    Y.Descriptor = D2

```

(a)

```

SELECT Title, Price
FROM   Books
WHERE  Descriptors  $\supseteq$  {D1, D2}
AND    Authors  $\supseteq$  {A2}

```

(b)

Figure 3-11. Formulation of query in (a) SQL referring to Figure 3-10, and (b) extended SQL referring to Figure 3-9.

itively, a simpler formulation should be possible, as indicated in Figure 3-11b.

In Chapter 8, we present an SQL-like extension for \rightarrow 1NF databases which makes possible queries like the one shown in Figure 3-11b.

Chapter 4

Formal Query Languages

In this chapter, we provide formal definitions for a tuple relational calculus and a relational algebra extended for the \neg 1NF model. The proof that these two formal languages are equivalent will be given in Chapter 6 after we have introduced some extended algebra operators in Chapter 5. These extended operators can be expressed in terms of the basic algebra operators and will simplify the proof development. Note that in this chapter we do not allow null values or empty nested relations. See Chapter 7 for a thorough treatment of null values.

4.1 Extended Relational Calculus

Using the notation from Chapter 2, we define a tuple relational calculus (TRC) with expressions of the form $\{t \mid \psi(t)\}$, where t is a tuple variable of fixed length and ψ is a formula built from atoms and a collection of operators defined below.

The atoms of formulas ψ are of four types.

1. $s \in r$, where s is a tuple variable, and r is a relation name. This specifies that s is a tuple in relation r , or s is an element of r . The arity of s is equal to the degree of r .
2. $s \in t[i]$ where t and s are tuple variables. This specifies that s is a

tuple in the relation specified by the i th component of t , whose value must be a set-of-tuples. The arity of s is the arity of the tuples in the set.

3. $a \theta s[i], s[i] \theta a, s[i] \theta t[j]$, where s and t are tuple variables, a is a constant, and θ is an arithmetic comparison operator ($=, >$). Note that constants may be simple values or non-empty sets-of-values, however equality is the only operator which can compare non-simple values. Although other comparison operators, such as $<, \supseteq, \subset$, etc., are legitimate operators and could be included in the calculus, for simplicity we use only $=$ and $>$. Expressions using these additional operators can be expressed with calculus expressions which do not use them.
4. $s[i] = \{u | \psi'(u, t_1, t_2, \dots, t_k)\}$, where ψ' is a formula with free tuple variables u, t_1, t_2, \dots, t_k ; s is some t_j . This specifies that the i th attribute of s is the set of u tuples such that ψ' holds. Note, if no tuples u satisfy ψ' then this atom evaluates to false. This is to comply with our requirement that no null values appear in instances.

As in Chapter 2, formulas are defined with the operators ($\neg, \wedge, \vee, \forall, \exists$).

To illustrate these concepts, let us consider a number of examples.

1. Given a 1NF relation r on scheme $R = (A, B)$, the TRC expression

which nests r on the B attribute producing a relation with scheme $R' = (A, B')$, $B' = (B)$ is:

$$\{t^{(2)} | (\exists s)(s \in r \wedge t[1]=s[1] \wedge t[2]=\{u^{(1)} | (\exists v)(v \in r \wedge s[1]=v[1] \wedge u[1]=v[2])\})\}$$

2. Given a nested relation r with scheme $R = (A, B')$, $B' = (B)$, the 1NF relation with scheme $R' = (A, B)$ is:

$$\{t^{(2)} | (\exists s)(s \in r \wedge t[1]=s[1] \wedge (\exists u)(u \in s[2] \wedge t[2]=u[1]))\}$$

3. Given a nested relation r with scheme $R = (A, B, E')$, $B = (C, D')$, $D' = (D)$, $E' = (E)$, the set of all tuples in r with a C value of 'c' and within that B tuple a D value of 'd', is:

$$\{t | t \in r \wedge (\exists s)(s \in t[2] \wedge s[1]='c' \wedge (\exists u)(u \in s[2] \wedge u[1]='d'))\}$$

4. Given a nested relation as in example 3, the set of all tuples in r , removing all B tuples from each B subrelation that do not have any D values greater than 6, and in those that do, eliminating all D values ≤ 6 , is:

$$\{t^{(3)} | (\exists s)(s \in r \wedge t[1]=s[1] \wedge t[3]=s[3] \wedge t[2]=\{u^{(2)} | (\exists v)(v \in s[2] \wedge u[1]=v[1] \wedge u[2]=\{w^{(1)} | w \in v[2] \wedge w[1] > 6\})\})\}$$

Figure 4-1 shows a sample relation r and the result of this query.

A	B		E'
	C	D'	E
		D	
a ₁	c ₁	11	e ₁
		2	e ₂
a ₁	c ₂	1	e ₃
		3	
a ₂	c ₁	11	e ₁
		14	e ₃
		6	
	c ₃	14	
		16	

A	B		E'
	C	D'	E
		D	
a ₁	c ₁	11	e ₁
			e ₂
a ₁	c ₁		e ₃
a ₂	c ₁	11	e ₁
		14	e ₃
	c ₃	14	
		16	

Figure 4-1. Relation r and result of calculus query 4.

As we pointed out in Chapter 2, the TRC allows us to define some infinite relations such as $\{t \mid \neg(t \in r)\}$, which denotes all possible tuples that are not in r , but are of the arity we associate with t . These types of expressions have not been eliminated in our present calculus and can even occur in nested expressions.

To overcome this problem, the notion of *safety* must be extended to the \neg 1NF calculus. *Safe* expressions are those expressions for which the answer can be computed in finite time by examining only the relations and constants mentioned in the expression. As for the 1NF calculus we denote the set of symbols that appear in relations or constants mentioned in expression ψ as $DOM(\psi)$. However, in the \neg 1NF calculus the symbols may appear also in nested relations. An expression ψ is safe if each component of any t that satisfies ψ must be a member of or, recursively, a relation on $DOM(\psi)$. This

statement replaces the first constraint listed under safety in Chapter 2. The second and third constraints are similarly modified so that the components of a tuple variable are recursively accessed, allowing the components of nested relations to be tested.

We add also a fourth constraint to the definition of safe expressions to eliminate the uncontrolled creation of powersets. This new constraint is necessary because we have introduced the new atom, $s \in t[i]$. This atom states that tuple variable s must assume values which are elements of the i th attribute of tuple variable t . Thus, $t[i]$, if not further constrained in the expression, can assume any set of values as long as a value for s is a member of that set. The first three safety constraints have only the capability of limiting the values for these sets to those in $DOM(\psi)$, the worst case being the powerset of $DOM(\psi)$. Our fourth constraint is as follows:

4. If an atom of the form $s \in t[i]$ appears in an expression then one of the following cases holds:
 - a. Tuple variable t appears in an atom of type $t \in r$.
 - b. Tuple variable t appears in an atom of type $t \in u[j]$.
 - c. The i th component of t appears in an atom of type $t[i] = u[j]$ or $u[j] = t[i]$ and if u appears in an atom of the form $q \in u[j]$ then safety constraint 4 is satisfied for u without considering

the atoms involving $t[i]$ which invoked this case.

- d. The i th component of t appears in an atom of type $t[i] = \{u|\psi'(u)\}$.

With this modification of $DOM(\psi)$ and the addition of constraint 4, and the proviso that *each* calculus expression, nested or otherwise, must be safe, our definition of safety for the $\neg 1NF$ calculus is complete.

4.2 Extended Relational Algebra

In order to have the same power as the safe relational calculus, we need to add only two new operators to the basic set of union, set difference, cartesian product, projection, and selection. These are the *nest* (ν) and *unnest* (μ) operators as defined in [JS, FT]. The basic set of operators work exactly as before except the domains may now be either atomic or set-valued.

1. *Nest* takes a relation structure $\mathcal{R} = \langle R, \tau \rangle$ and aggregates over equal data values in some subset of the names in R . Formally, let R be a relation scheme, in database scheme S , which contains a rule $R = (A_1, A_2, \dots, A_n)$ for external name R . Let $\{B_1, B_2, \dots, B_m\} \subset E_R$ and $\{C_1, C_2, \dots, C_k\} = E_R - \{B_1, B_2, \dots, B_m\}$. Assume that either the rule $B = (B_1, B_2, \dots, B_m)$ is in S or that B does not appear on the left hand side of any rule in S and (B_1, B_2, \dots, B_m) does not appear

on the right hand side of any rule in S . Then $\nu_{B=(B_1, B_2, \dots, B_m)}(\mathcal{R}) = \langle R', r' \rangle = \mathcal{R}'$ where:

$$1. R' = (C_1, C_2, \dots, C_k, (B_1, B_2, \dots, B_m)) = (C_1, C_2, \dots, C_k, B)$$

and the rule $B = (B_1, B_2, \dots, B_m)$ is appended to the set of rules in S if it is not already in S , and

$$2. r' = \{t \mid \text{there exists a tuple } u \in r \text{ such that}$$

$$t[C_1 C_2 \dots C_k] = u[C_1 C_2 \dots C_k] \wedge t[B] = \{v[B_1 B_2 \dots B_m] \mid v \in r \wedge v[C_1 C_2 \dots C_k] = t[C_1 C_2 \dots C_k]\}.$$

2. *Unnest* takes a relation structure nested on some set of attributes and disaggregates the structure to make it a “flatter” structure. Formally, let R be a relation scheme, in database scheme S , which contains a rule $R = (A_1, A_2, \dots, A_n)$ for external name R . Assume B is some higher order name in E_R with an associated rule $B = (B_1, B_2, \dots, B_m)$. Let $\{C_1, C_2, \dots, C_k\} = E_R - B$. Then $\mu_{B=(B_1, B_2, \dots, B_m)}(\mathcal{R}) = \langle R', r' \rangle = \mathcal{R}'$ where:

$$1. R' = (C_1, C_2, \dots, C_k, B_1, B_2, \dots, B_m) \text{ and the rule}$$

$B = (B_1, B_2, \dots, B_m)$ is removed from the set of rules in S if it does not appear in any other relation scheme, and

$$2. r' = \{t \mid \text{there exists a tuple } u \in r \text{ such that}$$

$$t[C_1 C_2 \dots C_k] = u[C_1 C_2 \dots C_k] \wedge t[B_1 B_2 \dots B_m] \in u[B].$$

Note that unnesting an empty set produces no tuples; however, since we do not allow empty nested relations and since the other algebra operators, in particular the nest operator, cannot produce them, there should be no need to apply unnest to an empty set.

We can apply unnest to a relation as long as it still contains nested relations. Thomas and Fischer [TF] showed that the order of unnesting does not affect the content of the resulting 1NF relation. They defined the UNNEST* operator to transform any \neg 1NF relation to a 1NF one. We will use μ^* to indicate this operation.

We often omit the right hand side of rules in unnest operations since the rule name is adequate. In a similar manner, when writing a nest operation we may choose not to specify the name of the rule to be added to S , only the name of the attributes to be nested. When this is done, we assume that a unique rule name is generated if the names being nested do not already appear on the right hand side of any rule in S .

Let us consider a number of examples to illustrate these concepts.

1. Given the relation r on scheme $R = (A, C, D, E)$, the relation with the C and D attributes nested together, and renamed B , is:

$$\nu_{B=(C,D)}(r)$$

This produces the scheme $R' = (A, B, E), B = (C, D)$.

A	C	D	E
a ₁	c ₁	d ₁	e ₁
a ₁	c ₁	d ₂	e ₁
a ₁	c ₁	d ₂	e ₂
a ₂	c ₂	d ₁	e ₁
a ₂	c ₂	d ₁	e ₂

A	B		E'
	C	D	E
a ₁	c ₁	d ₁	e ₁
	c ₁	d ₂	
a ₁	c ₁	d ₂	e ₂
a ₂	c ₂	d ₁	e ₁
a ₂	c ₂	d ₁	e ₂

A	B		E'
	C	D	E
a ₁	c ₁	d ₁	e ₁
a ₁	c ₁	d ₂	e ₁
a ₁	c ₁	d ₂	e ₂
a ₂	c ₂	d ₁	e ₁
a ₂	c ₂	d ₁	e ₂

Figure 4-2. Relation r and result of algebra query 2.

2. Using the same relation r , the relation with scheme $R' = (A, B, E')$, $B = (C, D)$, $E' = (E)$ is:

$$\nu_{B=(C,D)}(\nu_{E'=(E)}(r)) \quad \text{or} \quad \nu_{E'=(E)}(\nu_{B=(C,D)}(r))$$

Although both of these expressions produce the desired scheme, the relations may be radically different (see Figure 4-2).

3. The relation on scheme $R' = (A, B, E)$, $B = (C, D')$, $D' = (D)$ produced from r is:

$$\nu_{B=(C,D')}(\nu_{D'=(D)}(r))$$

In this case only one order is possible since D must be nested before D' can be further nested as part of B .

4. Given the relation s on $S = (A, B, E')$, $B = (C, D)$, $E' = (E)$, the relation with attribute E' unnested, is:

$$\mu_{E'}(s)$$

5. Given relation s on S as in 4, the relation with attribute B unnested,

giving the scheme $S' = (A, C, D, E')$, $E' = (E)$, is:

$$\mu_B(s)$$

6. Given relation s on S as in 4, the relation with each of the D' sets within each B subrelation unnested, producing the relation with

scheme $S' = (A, B, E')$, $B = (C, D)$, $E' = (E)$, is:

$$\nu_{B=(C,D)}(\mu_{D'}(\mu_B(s)))$$

Chapter 5

Partitioned Normal Form and Extended Algebra Operators

In this chapter, we consider a restriction of 1NF relations to those that are in *partitioned normal form*. We then define a set of extended algebra operators under which the class of partitioned normal form relations is closed. These extended operators are designed to be *reasonable* extensions to their 1NF counterparts, making use of the implied multivalued dependencies which exist when relations are in partitioned normal form.

5.1 Restricting the Class of 1NF Relations

Consider the relation scheme

Student = (sname, Course)

Course = (cname, grade)

In Figure 5-1 we have two instances of Student, S_1 and S_2 , where S_1 contains previous work of two students and S_2 contains some new data on these students.

A natural step would be to add the new information in S_2 to that in S_1 . If we apply the union operator then we get the relation in Figure 5-2.

Although all of the information is certainly represented in this relation it lacks the intuitive appeal of the relation in Figure 5-3 in which the Course sets are combined for each unique value of Student. One alternative is to use

sname	Course	
	cname	grade
Jones	Math	A
	Science	B
Smith	Math	A
	Physics	C
	Science	A

sname	Course	
	cname	grade
Jones	Physics	B
Smith	Chemistry	A
	English	B

Figure 5-1. Two Student instances.

sname	Course	
	cname	grade
Jones	Math	A
	Science	B
Jones	Physics	B
Smith	Math	A
	Physics	C
	Science	A
Smith	Chemistry	A
	English	B

Figure 5-2. Union of instances in Figure 5-1.

an unnest operation followed by the corresponding nest operation after taking the union. So the query would be

$$\nu_{\text{Course}}(\mu_{\text{Course}}(S_1 \cup S_2))$$

This takes advantage of the property that, in general, nest is not always an inverse operator for unnest. This property is intuitively unappealing and impedes query optimization.

sname	Course	
	cname	grade
Jones	Math	A
	Science	B
	Physics	B
Smith	Math	A
	Physics	C
	Science	A
	Chemistry	A
	English	B

Figure 5-3. Better representation of Figure 5-2.

We, therefore, define a class of 1NF relations for which there is always a sequence of nest operations which will be an inverse for any sequence of valid unnest operations. In the next section, we extend the meaning of our relational algebra operators to work within this domain.

Definition 5.1: Let $\mathcal{R} = \langle R, r \rangle$ be a relation structure with attribute set E_R containing zero order attributes A_1, A_2, \dots, A_k and higher order attributes X_1, X_2, \dots, X_ℓ . \mathcal{R} is in *partitioned normal form* (PNF) if and only if the following two conditions hold:

- (a) $A_1 A_2 \dots A_k \rightarrow E_R$, and
- (b) For all $t \in r$ and for all $X_i : 1 \leq i \leq \ell$: \mathcal{R}_{ti} is in PNF, where $\mathcal{R}_{ti} = \langle X_i, t[X_i] \rangle$.

Note, if $k = 0$ then $\emptyset \rightarrow E_R$ must hold and if $\ell = 0$ then $A_1 A_2 \dots A_k \rightarrow A_1 A_2 \dots A_k$ holds trivially. Thus a 1NF relation is in PNF.

PNF is a desirable goal for the representation of relationships in $\neg 1NF$ relations. This stems from our belief that a particular nesting scheme should not be used unless the FDs which enforce PNF hold in the relation. We will discuss further normalization for $\neg 1NF$ relations in Chapter 9.

We would like to ensure that given a relation in PNF when we apply a nest or an unnest operator then we get a PNF relation in return. In general this is true only for the unnest operator. The nest operator returns a PNF relation if and only if certain functional dependencies hold in the relation and each nested relation.

Theorem 5-1. *The class of PNF relations is closed under unnesting.*

Proof: Let \mathcal{R} be any relation structure $\mathcal{R} = \langle R, r \rangle$ with attribute set E_R containing higher order attribute B with scheme $B = (B_1, B_2, \dots, B_q)$. We show that $\mathcal{R}' = \mu_{B=(B_1, B_2, \dots, B_q)} \mathcal{R}$ is a PNF relation.

Since \mathcal{R} is in PNF we know that $A_1 A_2 \cdots A_n \rightarrow E_R$ where the A_i , $1 \leq i \leq n$, are the zero order attributes in E_R . We also have that in each nested relation B , $B_1 B_2 \cdots B_\ell \rightarrow E_B$ where the B_i , $1 \leq i \leq \ell$, are the zero order attributes in E_B .

The attributes of \mathcal{R}' are, by definition of unnest, the attributes $(E_R - B) \cup (B_1 B_2 \cdots B_n)$. These attributes can be partitioned into four sets, the zero order attributes of E_R ($A_1 A_2 \cdots A_n$), the higher order attributes in $E_R - B$

$(X_1X_2 \cdots X_m)$, the zero order attributes of E_B ($B_1B_2 \cdots B_\ell$), and the higher order attributes of E_B ($Y_1Y_2 \cdots Y_p$). Our task then is to show that for any tuples t_1 and t_2 , if $t_1[A_1A_2 \cdots A_nB_1B_2 \cdots B_\ell] = t_2[A_1A_2 \cdots A_nB_1B_2 \cdots B_\ell]$ then $t_1[X_1X_2 \cdots X_mY_1Y_2 \cdots Y_p] = t_2[X_1X_2 \cdots X_mY_1Y_2 \cdots Y_p]$.

Since $A_1A_2 \cdots A_n \rightarrow X_1X_2 \cdots X_m$ in \mathcal{R} , and unnesting only duplicates these values, we have that $t_1[X_1X_2 \cdots X_m] = t_2[X_1X_2 \cdots X_m]$. Since t_1 and t_2 agree on $A_1A_2 \cdots A_n$, they came from the same tuple of r , and in this tuple $B_1B_2 \cdots B_\ell \rightarrow Y_1Y_2 \cdots Y_p$. So in the set of tuples obtained after unnesting the same FD applies and since t_1 agrees with t_2 on $B_1B_2 \cdots B_\ell$, $t_1[Y_1Y_2 \cdots Y_p] = t_2[Y_1Y_2 \cdots Y_p]$. \square

Theorem 5-2. *The nesting of a PNF relation is in PNF iff in the PNF relation $\mathcal{R} = \langle R, r \rangle$, $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$, where A_1, A_2, \dots, A_k are the zero order attributes in E_R not being nested and X_1, X_2, \dots, X_ℓ are the higher order attributes in E_R not being nested.*

Proof: We show that $\mathcal{R}' = \nu_{X_0=(A_{k+1}, A_{k+2}, \dots, A_n, X_{\ell+1}, X_{\ell+2}, \dots, X_m)}(\mathcal{R})$ is in PNF if and only if $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$, where A_1, A_2, \dots, A_n are the zero order attributes in E_R and X_1, X_2, \dots, X_m are the higher order attributes in E_R .

if: We prove that, if $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$ then \mathcal{R}' is in PNF. We utilize a case analysis on the values on m, n, k , and ℓ . Note that either $k < n$ or $\ell < m$ if we are nesting something.

Case 1: $m = 0, n > 0$. Then we have a 1NF relation and by definition of nest the relation is partitioned by the nonnested attributes $A_1A_2 \cdots A_k$. So $A_1A_2 \cdots A_k \rightarrow X_0$ in \mathcal{R}' and thus \mathcal{R}' is in PNF.

Case 2: $m > 0, n = 0$. Then there is one tuple in the relation as the FD $\emptyset \rightarrow X_1X_2 \cdots X_m$ holds. Nesting cannot produce fewer tuples and any nested relation created can only have one tuple so the new relation is in PNF.

Case 3: $m > 0, n > 0, k < n, \ell = m$. Then we are nesting only zero order attributes. So $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_m$. Then in each partition on $A_1A_2 \cdots A_k$ the $X_1X_2 \cdots X_m$ values will be the same so a partition on $A_1A_2 \cdots A_kX_1X_2 \cdots X_m$, used by the nest, will be isomorphic to a partition on $A_1A_2 \cdots A_k$. The nest will form a set X_0 of $A_{k+1}A_{k+2} \cdots A_n$ values in each partition and the FD $A_1A_2 \cdots A_kX_1X_2 \cdots X_m \rightarrow X_0$ will hold. So $A_1A_2 \cdots A_k \rightarrow X_0X_1X_2 \cdots X_m$, giving a relation in PNF.

Case 4: $m > 0, n > 0, k = n, \ell < m$. Then we are nesting only higher order attributes. So $A_1A_2 \cdots A_n \rightarrow X_1X_2 \cdots X_\ell$. Nesting will be done by grouping $X_{\ell+1}X_{\ell+2} \cdots X_m$ in each tuple, since $A_1A_2 \cdots A_n$ will continue to form a tuple-wise partition. So $A_1A_2 \cdots A_n \rightarrow X_0X_1X_2 \cdots X_\ell$, giving a relation in PNF.

Case 5: $m > 0, n > 0, k < n, \ell < m$. Then we are nesting some zero order and some higher order attributes. So $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$. Then

during nesting a partition on $A_1A_2 \cdots A_k X_1X_2 \cdots X_\ell$ will be created and by definition each set X_0 of $A_{k+1}A_{k+2} \cdots A_n X_{\ell+1}X_{\ell+2} \cdots X_m$ values will be uniquely determined by $A_1A_2 \cdots A_k X_1X_2 \cdots X_\ell$. Thus, $A_1A_2 \cdots A_k \rightarrow X_0X_1X_2 \cdots X_\ell$. In each new nested relation the $A_{k+1}A_{k+2} \cdots A_n$ values are unique since $A_1A_2 \cdots A_k$ was the same for each of these tuples and $A_1A_2 \cdots A_n$ values were unique as \mathcal{R} is in PNF. Thus $A_{k+1}A_{k+2} \cdots A_n \rightarrow X_{\ell+1}X_{\ell+2} \cdots X_m$ in each nested relation. Thus the relation is in PNF.

only if: We prove if \mathcal{R}' is in PNF then $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$.

Since $A_1A_2 \cdots A_k$ are the zero order attributes of \mathcal{R}' , by definition of PNF $A_1A_2 \cdots A_k \rightarrow E_{\mathcal{R}'}$ holds in \mathcal{R}' . By the projectivity FD axiom, $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$.

Therefore, \mathcal{R}' is in PNF iff $A_1A_2 \cdots A_k \rightarrow X_1X_2 \cdots X_\ell$. □

5.2 Extending the Basic Relational Algebra Operators

As the example in section 5-1 showed, we need to extend our basic algebra operators to work within the class of PNF relations. We first extend the traditional set operators—union, intersection, difference, and cartesian product, and then extend natural join and projection. Some of these operators are similar to the extended operators of [AB2]. However, our definitions arose out of the PNF requirement and since our model does not include null values or empty sets, the

operations are well defined. In [AB2], empty sets are allowed but null values are not, so there are problems when tuples with empty sets are unnested. Unlike [AB2], we do not extend selection in this dissertation. We note also that the extended operators can be applied to non-PNF relations in a well defined way, however, the result is not necessarily a PNF relation.

We find that there is not much correspondence between the way most of the relational algebra operators work on 1NF relations and their counterpart \neg 1NF relations.

Example 5.1: Consider \neg 1NF relations r_1 and r_2 of Figure 5-4 and their 1NF counterparts, s_1 and s_2 . Note, however, that $r_1 \cap r_2$ is not the \neg 1NF counterpart of $s_1 \cap s_2$, as the usual definition of intersection requires that a tuple is in the result only if that tuple is in both input relations. \square

We believe that each 1NF operator should have a *reasonable* \neg 1NF counterpart. Intuitively, a \neg 1NF operator is *reasonable* if it behaves identically to the corresponding 1NF operator on 1NF relations and if it produces a result which would have been produced had the equivalent set of 1NF relations been used instead of \neg 1NF relations. We now formally define *reasonable* in terms of faithfulness and precision.

Let Rel be the set of all 1NF relations and let Rel^* be the set of all \neg 1NF relations that have at least one higher order attribute in the scheme. Thus, $Rel \cap Rel^* = \emptyset$.

r_1	r_2	$r_1 \cap r_2$																																
<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B*</td></tr> <tr><td></td><td>B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td></td><td>b'</td></tr> <tr><td>a'</td><td>b</td></tr> <tr><td></td><td>b'</td></tr> </table>	A	B*		B	a	b		b'	a'	b		b'	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B*</td></tr> <tr><td></td><td>B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td></td><td>b'</td></tr> <tr><td>a'</td><td>b</td></tr> <tr><td></td><td>b''</td></tr> </table>	A	B*		B	a	b		b'	a'	b		b''	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B*</td></tr> <tr><td></td><td>B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td></td><td>b'</td></tr> </table>	A	B*		B	a	b		b'
A	B*																																	
	B																																	
a	b																																	
	b'																																	
a'	b																																	
	b'																																	
A	B*																																	
	B																																	
a	b																																	
	b'																																	
a'	b																																	
	b''																																	
A	B*																																	
	B																																	
a	b																																	
	b'																																	
s_1	s_2	$s_1 \cap s_2$																																
<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td>a</td><td>b'</td></tr> <tr><td>a'</td><td>b</td></tr> <tr><td>a'</td><td>b'</td></tr> </table>	A	B	a	b	a	b'	a'	b	a'	b'	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td>a</td><td>b'</td></tr> <tr><td>a'</td><td>b</td></tr> <tr><td>a'</td><td>b''</td></tr> </table>	A	B	a	b	a	b'	a'	b	a'	b''	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 50px;">A</td><td style="width: 50px;">B</td></tr> <tr><td>a</td><td>b</td></tr> <tr><td>a</td><td>b'</td></tr> <tr><td>a'</td><td>b</td></tr> </table>	A	B	a	b	a	b'	a'	b				
A	B																																	
a	b																																	
a	b'																																	
a'	b																																	
a'	b'																																	
A	B																																	
a	b																																	
a	b'																																	
a'	b																																	
a'	b''																																	
A	B																																	
a	b																																	
a	b'																																	
a'	b																																	

Figure 5-4. Intersection applied to $\neg 1NF$ and $1NF$ relations.

Definition 5.2: Let γ be an operator on Rel and let γ' be an operator on $Rel^* \cup Rel$. We say that γ' is *faithful* to γ if one of the following two conditions holds:

1. when γ and γ' are unary operators, $\gamma(r) = \gamma'(r)$ for every $r \in Rel$ for which $\gamma(r)$ is defined.
2. when γ and γ' are binary operators, $r \gamma q = r \gamma' q$ for every $r, q \in Rel$ for which $r \gamma q$ is defined.

Definition 5.3: Let γ be an operator on Rel and let γ' be an operator on Rel^* . We say that γ' is a *precise generalization* of γ relative to unnesting if one of the following two conditions holds:

1. when γ and γ' are unary operators, $\mu^*(\gamma'(r)) = \gamma(\mu^*(r))$ for every

$r \in Rel^*$ for which $\gamma'(r)$ is defined.

2. when γ and γ' are binary operators, $\mu^*(r \gamma' q) = \mu^*(r) \gamma \mu^*(q)$ for every $r, q \in Rel^*$ for which $r \gamma' q$ is defined.

We now define $\neg 1NF$ operators which are faithful and precise and also have some intuition behind them.

5.2.1 Extended Union

In order to take the *extended union* of two relations r_1 and r_2 we require that they have equal relation schemes, say R . The scheme of the resultant structure is also equal to R . We define *extended union* at the instance level as follows.

Definition 5.4: Let r_1 and r_2 be relations on scheme R . Let X range over the zero order attributes in E_R and Y range over the higher order attributes in E_R . The *extended union* of r_1 and r_2 is:

$$\begin{aligned}
 r_1 \cup^e r_2 = \{t \mid & (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 : (\forall X, Y \in E_R : t[X] = t_1[X] = t_2[X] \\
 & \wedge t[Y] = (t_1[Y] \cup^e t_2[Y]))) \\
 \vee & (t \in r_1 \wedge (\forall t' \in r_2 : (\forall X \in E_R : t[X] \neq t'[X]))) \\
 \vee & (t \in r_2 \wedge (\forall t' \in r_1 : (\forall X \in E_R : t[X] \neq t'[X])))\}
 \end{aligned}$$

Note, this definition is recursive in that we apply the *extended union* to each higher order attribute Y .

A	B*	C*
	B	C
a	b	c
		c'

A	B*	C*
	B	C
a	b'	c

A	B*	C*
	B	C
a	b	c
	b'	c'

A	B	C
a	b	c
a	b	c'
a	b'	c
a	b'	c'

A	B	C
a	b	c
a	b	c'

A	B	C
a	b'	c

A	B	C
a	b	c
a	b	c'
a	b'	c

Figure 5-5. Counterexample to preciseness of \cup^e .

Proposition 5.1: Extended union is *faithful* to standard union.

Proof: The definition of \cup^e differs from the definition of \cup only when higher order attributes are present in the scheme. When there are no higher order attributes, as in *Rel*, then the definition of \cup^e reduces to a selection of tuples that are in both relations or are tuples in only one of the two relations, i.e., a standard union. □

Proposition 5.2: Extended union is *not a precise* generalization of standard union with respect to unnesting.

Proof: Figure 5-5 shows two $\neg 1NF$ relations r_1 and r_2 where $\mu^*(r_1 \cup^e r_2) \neq \mu^*(r_1) \cup \mu^*(r_2)$. □

Extended union is not precise due to the syntactic nature of standard

union. Standard union does not take into account dependencies that should exist in a relation if it is going to be nested. If we agree that only relations from Rel^* which are in PNF should be allowed, then each nesting scheme is allowed if and only if certain multivalued dependencies hold in the completely unnested relation.

The intuition behind the extended union, and, as we will see, the other extended operators, is to take advantage of the MVDs which allow us to nest relations and maintain partitioned normal form. For instance, in the example of Figure 5-5 the MVD $A \twoheadrightarrow B \mid C$ holds. Thus, B and C values are only indirectly related through the A attribute, and the primary associations are AB and AC . Since we can think of union as an insertion operation, we would like to be able to insert AB and AC associations independently of each other.

With a 1NF relation on ABC this is not possible unless we specify every existing AC association for an A -value whenever we add a new AB association for that A value, and vice versa. However, in the \neg 1NF relation each A value functionally determines a B^* and C^* set, and each set can be independently updated with our extended union. A similar result can be achieved by decomposing each ABC relation into AB and AC , which is the *path set* for this scheme (see section 3.4). We then perform a standard union among the corresponding decomposed relations, and finally rejoin. Proposition 3.1 ensures that the same MVDs will hold in the new result and so the same nesting structure

will be possible.

If we use a modified version of standard union which takes into account the MVDs or, equivalently, the join dependency which produces the nested structure, then we have a *precise* extended union operator.

Definition 5.5: Let $\bowtie (X_1, X_2, \dots, X_n)$ be a join dependency on scheme R with zero order attributes $E_R = X_1 \cup X_2 \cup \dots \cup X_n$. The *decomposition union* (or Δ -union) of two 1NF relations r_1 and r_2 on R is

$$r_1 \cup^\Delta r_2 = \bowtie (r_1[X_1] \cup r_2[X_1], r_1[X_2] \cup r_2[X_2], \dots, r_1[X_n] \cup r_2[X_n])$$

where \bowtie is the standard natural join.

Proposition 5.3: Extended union is a *precise* generalization of Δ -union with respect to unnesting, where the join dependency used in the Δ -union is the path set of the 1NF relation's scheme tree.

Proof: We need to show that $\mu^*(r) \cup^\Delta \mu^*(q) = \mu^*(r \cup^e q)$ for any $r, q \in Rel^*$ for which $r \cup^e q$ is defined, i.e., r and q have identical relation schemes. We show inclusion both ways to prove the equivalence.

\subseteq Let t be a tuple in $\mu^*(r) \cup^\Delta \mu^*(q)$. Two cases need to be considered: either t came only from tuples in one of $\mu^*(r)$ or $\mu^*(q)$, or t is a combination of tuples from $\mu^*(r)$ and $\mu^*(q)$, put together via the join operation in the Δ -union.

Case 1: Suppose t came directly from tuples in $\mu^*(r)$. The argument

for q is symmetrical. Due to the join dependency holding in $\mu^*(r)$, all of these tuples agree on the join attributes which are the non-leaf nodes in the scheme tree for r . Thus, we know that there is one tuple in $\mu^*(r)$ which decomposed and rejoined to make t . This tuple unnested from a single tuple t_r in r . Now any tuple in r must either be intact in $r \cup^e q$ if there was no tuple in q with the same partition key, or there is some tuple t' in $r \cup^e q$ in which each nested relation of t_r is a subset of the corresponding nested relation in t' . In either case, unnesting $r \cup^e q$ will return the original tuple t .

Case 2: If t was created by taking pieces of tuples from both $\mu^*(r)$ and $\mu^*(q)$, as in Case 1, the tuples from which it came must agree on the non-leaf nodes in the scheme tree for r and q . Thus the tuples from r and q which unnested to these tuples interact in the extended union of r and q which, when unnested, must contain the tuple t .

\supseteq Let T be a set of tuples in $\mu^*(r \cup^e q)$ such that all tuples in T unnested from a single tuple t in $r \cup^e q$. Two cases need to be considered: either t comes only from r or q , or t is a combination of tuples in t_r in r and t_q in q .

Case 1: Suppose t came only from r . The argument for q is symmetrical. All tuples in T will get decomposed and rejoined by the Δ -union, plus perhaps participating with other tuples in the join. But at least

the original tuples will be returned, so all tuples in T are in the left hand side.

Case 2: Each tuple in T may take some of its values from attributes in t_r or t_q , but if the values of some attributes are different, then the attributes which are above that attribute in the scheme tree have equal values. This is exactly how the unnested tuples of t_r and t_q will interact in the join operation of the Δ -union. So every tuple in T will be the join of pieces from an unnested tuple t_r of r and an unnested tuple t_q of q . □

We note that Proposition 5.3 gives us a method for expressing extended union in terms of the basic algebra operators. The operands must have known schemes and so we can use the path set of the associated scheme tree to perform the projection involved in the decomposition union. The sequence of operations would be to completely unnest each operand, project using the components determined by the path set, union each of the corresponding components of each operand, join the new components (using select and cartesian product), and nest to gain the original structure. If the operands were not in PNF, then appropriate algebra operators could be used to add a key to each relation or nested relation so that the relations are in PNF (see Chapter 6), the above procedure applied, and then the keys removed.

5.2.2 Extended Intersection

Extended intersection has the same scheme requirements as extended union. Two tuples intersect if they agree on their zero order attributes and they have non-empty extended intersections of their higher order attributes. Since we do allow empty nested relations to appear in the $\neg 1NF$ model without null values, a tuple with an empty extended intersection of some higher order attributes must be eliminated. This is also critical if extended intersection is to be a precise generalization of standard intersection.

Definition 5.6: Let r_1 and r_2 be relations on scheme R . Let X range over the zero order attributes in E_R and Y range over the higher order attributes in E_R . The *extended intersection* of r_1 and r_2 is:

$$r_1 \cap^e r_2 = \{t \mid (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 : (\forall X, Y \in E_R : t[X] = t_1[X] = t_2[X] \\ \wedge t[Y] = (t_1[Y] \cap^e t_2[Y]) \wedge t[Y] \neq \emptyset))\}$$

Proposition 5.4: Extended intersection is *faithful* to standard intersection.

Proof: As in the proof for union, the definition of \cap^e differs from the definition of \cap only when higher order attributes are in the scheme. When only relations in *Rel* are being considered, the definition of \cap^e reduces to the definition of standard intersection. □

Proposition 5.5: Extended intersection is a *precise* generalization of standard intersection with respect to unnesting.

Proof: We need to show that $\mu^*(r) \cap \mu^*(q) = \mu^*(r \cap^e q)$ for any $r, q \in Rel^*$ for which $r \cup^e q$ is defined. We show inclusion both ways to prove the equivalence.

\subseteq Let t be a tuple in $\mu^*(r) \cap \mu^*(q)$. Then, $t \in \mu^*(r)$ and $t \in \mu^*(q)$. Now, t unnested from some tuple $t_r \in r$ and some tuple $t_q \in q$. Furthermore, t_r and t_q agree on the attributes which are the non-leaf nodes in the scheme tree for r and q . Therefore, when $r \cap^e q$ is calculated, t_r and t_q will participate in the result, and when unnested, will produce the tuple t . Thus, $t \in \mu^*(r \cap^e q)$.

\supseteq Let T be a set of tuples in $\mu^*(r \cap^e q)$ such that all tuples in T unnested from a single tuple t in $r \cap^e q$. Then, all tuples in T agree with t on the attributes which are the non-leaf nodes in the scheme tree for r and q . Furthermore, the only values of attributes which are leaf nodes, which are in tuples of T , are those that were in both the r and q tuples which participated to form t . Thus, a tuple is in T exactly when it agrees with some tuple unnested from r and some tuple unnested from q . That is, $\forall t' \in T : t' \in \mu^*(r) \cap \mu^*(q)$. \square

We note that a Δ -intersection operator could be defined in a similar manner to Δ -union, although it is not necessary as $r_1 \cap r_2 = r_1 \cap^\Delta r_2$ for any $r_1, r_2 \in Rel$. Also, the comments made about expressing extended union in terms of the basic relational algebra operators can also be applied to extended intersection except that the decomposition and join steps are not required in

the transformation.

5.2.3 Extended Difference

The *extended difference* operator has semantic complications similar to extended union. *Extended difference* also has the same scheme requirements as union. In $r_1 -^e r_2$ a tuple is retained from r_1 if it does not agree with any tuple in r_2 on the zero order attributes or if it does then it has non-empty extended differences between the higher order attributes. Our comments on empty nested relations from section 5.2.2 apply here as well.

Definition 5.7: Let r_1 and r_2 be relations on scheme R . Let X range over the zero order attributes in E_R and Y and Z range over the higher order attributes in E_R . The *extended difference* of r_1 and r_2 is:

$$r_1 -^e r_2 = \{t \mid (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 \wedge \exists Z \in E_R : (\forall X, Y \in E_R : \\ t[X] = t_1[X] = t_2[X] \wedge t[Y] = (t_1[Y] -^e t_2[Y]) \wedge t[Y] \neq \emptyset) \\ \vee (t \in r_1 \wedge (\forall t' \in r_2 : (\forall X \in E_R : t[X] \neq t'[X])))\})\}$$

Proposition 5.6: Extended difference is *faithful* to standard difference.

Proof: Similar to proofs for union and intersection. □

Proposition 5.7: Extended difference is *not* a *precise* generalization of standard difference with respect to unnesting.

Proof: Figure 5-6 shows two 1NF relations r_1 and r_2 where $\mu^*(r_1 -^e r_2) \neq \mu^*(r_1) - \mu^*(r_2)$. □

A	B*	C*
	B	C
a	b	c
		c'

A	B*	C*
	B	C
a	b'	c

A	B*	C*
	B	C
a	b	c'

A	B	C
a	b	c'

A	B	C
a	b	c
a	b	c'

A	B	C
a	b'	c

A	B	C
a	b	c
a	b	c'

Figure 5-6. Counterexample to preciseness of $-^e$.

The intuition behind this definition of *extended difference* is similar to the intuition behind extended union. We think of difference as the deletion of information from the database. In the counterexample in Figure 5-6, we are trying to delete two relationships from r_1 , the AB association between a and b' and the AC association between a and c. Since there is no association between a and b' in r_1 , nothing changes due to that request. However, the a to c association is in r_1 and so it is removed. In the 1NF versions of r_1 and r_2 , it is not possible to express only an AB or an AC relationship, but only an artificial ABC relationship. Thus in order to delete, say, an AC association, we would have to know all of the B values associated with the A value so all ABC relationships could be deleted.

As with union, the problem stems from the MVDs that must exist in the 1NF counterparts of the \neg 1NF relations. Our solution follows the same line as for union. We first decompose the relation via the join dependency specified

by the scheme tree, perform the difference on the decomposed relations and then rejoin.

Definition 5.8: Let $\bowtie (X_1, X_2, \dots, X_n)$ be a join dependency on scheme R with zero order attributes $E_R = X_1 \cup X_2 \cup \dots \cup X_n$. The *decomposition difference* or Δ -*difference*, of two 1NF relations r_1 and r_2 on R is

$$r_1 -^\Delta r_2 = \bowtie (r_1[X_1] - r_2[X_1], r_1[X_2] - r_2[X_2], \dots, r_1[X_n] - r_2[X_n])$$

where \bowtie is the natural join.

Proposition 5.8: Extended difference is a *precise* generalization of Δ -difference with respect to unnesting, where the join dependency used in the Δ -difference is the path set of the \neg 1NF relation's scheme tree.

Proof: We need to show that $\mu^*(r) -^\Delta \mu^*(q) = \mu^*(r -^e q)$ for any $r, q \in Rel^*$ for which $r -^e q$ is defined, i.e., r and q have identical relation schemes. We show inclusion both ways to prove the equivalence.

\subseteq Let t be a tuple in $\mu^*(r) -^\Delta \mu^*(q)$. Two cases need to be considered: either t came only from tuples in $\mu^*(r)$, or t is a combination of tuples from $\mu^*(r)$ and $\mu^*(q)$, put together via the join operation in the Δ -difference.

Case 1: Suppose t came directly from tuples in $\mu^*(r)$. Due to the join dependency holding in $\mu^*(r)$, all of these tuples agree on the join attributes which are the non-leaf nodes in the scheme tree for r .

Thus, we know that there is one tuple in $\mu^*(r)$ which decomposed and rejoined to make t . This tuple unnested from a single tuple t_r in r . Since t came directly from $\mu^*(r)$, it was not affected by tuples in q . So $t_r \in r -^e q$, and unnesting $r -^e q$ will return the original tuple t .

Case 2: If t was created by taking pieces of tuples from $\mu^*(r)$ that were not in $\mu^*(q)$, as in Case 1, the tuples from which it came must agree on the non-leaf nodes in the scheme tree for r and q . Thus the tuples from r and q which unnested to these tuples interact in the extended difference of r and q which, when unnested, must contain the tuple t .

\supseteq Let T be a set of tuples in $\mu^*(r -^e q)$ such that all tuples in T unnested from a single tuple t in $r -^e q$. Two cases need to be considered: either t comes only from r , or t is a combination of tuples in t_r in r and t_q in q .

Case 1: Suppose t came only from r . All tuples in T will get decomposed and rejoined by the Δ -difference. Thus, the original tuples will be returned, so all tuples in T are in the left hand side.

Case 2: Each tuple in T may take some of its values from attributes in t_r that are not in t_q , but only if the attributes which are above that attribute in the scheme tree have equal values. This is exactly how the unnested tuples of t_r and t_q will interact in the join operation of

the Δ -difference. So every tuple in T will be the join of pieces from an unnested tuple t_r of r that are not in an unnested tuple t_q of q . \square

We note that a procedure similar to that used to express extended union in terms of the basic relational algebra operators can be applied to extended difference.

5.2.4 Cartesian Product and Select

The standard product and select operators can be used on \neg 1NF relations. Since nest is an inverse for unnest when dealing with PNF relations, when products or selections on tuples within nested relations are desired, the appropriate attributes can be unnested, the operation performed, and the relation re-nested according to the user's desires.

More sophisticated predicates for *select* could be defined using set comparison operators (see [AB2, Sch1]), however these operators do not have a simple mapping to standard select. In fact set comparisons in the standard algebra usually require a combination of product, select, and project operators. There is a proposal for a recursive algebra [Jae3] in which the standard operators are applied to nested relations in recursively constructed queries. These extensions appear to be precise generalizations, however a recursive algebra is beyond the scope of this dissertation.

5.2.5 Extended Natural Join

Join operations are difficult to define in the \neg 1NF model due to the possibility of different nesting depths for the attributes. The problems with an extended natural join (\bowtie^e) can be illustrated as follows.

Let r_1 be a relation on $R_1 = (A, X)$, $X = (B, C)$ and let r_2 be a relation on $R_2 = (B, D)$. Then $r_1 \bowtie^e r_2$ is the cartesian product of r_1 and r_2 since $E_{R_1} \cap E_{R_2} = \emptyset$. However, in the 1NF counterparts of r_1 and r_2 , attribute B is a common attribute so a join on B must take place. Thus, we limit the relations which can participate in an extended natural join to those whose only common attributes are elements of the top level scheme, i.e., in E_R for scheme R , or are attributes of a common higher order attribute. With a recursive algebra as discussed above, more general join operations could be defined.

Let r_1 be a relation on scheme R_1 and r_2 a relation on scheme R_2 . We define the *extended natural join* $r_1 \bowtie^e r_2$ as a recursive application of a rule similar to the definition of natural join used for standard 1NF relations.

In the standard natural join, two tuples contribute to the join if they agree on the attributes in common to both schemes. Under extended natural join, two tuple contribute to the join if the extended intersection of their projections over common attributes is not empty.

Definition 5.9: Let X be the higher order attributes in $E_{R_1} \cap E_{R_2}$, $A =$

$E_{R_1} - X$, and $B = E_{R_2} - X$. Then the *extended natural join* of r_1 and r_2 is

$r_1 \bowtie^e r_2$ which produces a relation r on scheme R where:

1. $R = (A, X, B)$, and
2. $r = \{t \mid (\exists u \in r_1, v \in r_2 : t[A] = u[A] \wedge t[B] = v[B] \wedge t[X] = (u[X] \cap^e v[X]) \wedge t[X] \neq \emptyset)\}$

Proposition 5.9: Extended natural join is *faithful* to standard natural join.

Proof: If there are no higher order attributes, then X is empty, and the definition of extended natural join reduces to the definition of standard natural join. □

Proposition 5.10: Extended natural join is *precise* generalization of standard natural join with respect to unnesting.

Proof: We need to show that $\mu^*(r) \bowtie \mu^*(q) = \mu^*(r \bowtie^e q)$ for any $r, q \in Rel^*$ for which $r \bowtie^e q$ is defined. We show inclusion both ways to prove the equivalence.

⊆ Let t be a tuple in $\mu^*(r) \bowtie \mu^*(q)$. Then, t agrees on all zero order attributes common to r and q and all attributes which unnested from common higher order attributes in r and q . Let $t_r \in r$ and $t_q \in q$, be the tuples that unnested to participate in producing t . In $r \bowtie^e q$, we will take the extended intersection of the common higher order attributes of r and q , producing only those values common to both. Since t_r and t_q agree on all attributes which unnest from the common higher order attributes, they will participate in the extended intersec-

tion, and when we unnest this result, the tuple t will appear.

\supseteq Let T be a set of tuples in $\mu^*(r \bowtie^e q)$ such that all tuples in T unnested from a single tuple t in $r \bowtie^e q$. Then, there are tuple $t_r \in r$ and $t_q \in q$ that participated to make t . These tuples agree on the common zero order attributes of r and q . Furthermore, t contains only values in the common higher order attributes that are in both t_r and t_q . Thus, when we unnest t_r and t_q and join the result we match up only on those same common values. Thus, all tuples of T are also in $\mu^*(r) \bowtie \mu^*(q)$.

□

5.2.6 Extended Projection

Extended projection is a normal projection followed by a tuplewise extended union of the result. The union merges tuples which agree on the zero order attributes left in the projected relation.

Definition 5.10: The *extended projection* of relation r on attributes X is

$$\pi_X^e(r) = \bigcup_{t \in \pi_X(r)} (t)$$

Note, that projection still removes duplicate tuples, that is those which agree on all attributes, with set equality holding on higher order attributes.

Proposition 5.11: Extended projection is *faithful* to standard projection.

Proof: When there are no higher order attributes, a tuple wise extended union of $\pi_X(r)$ will not add or delete any values. □

Proposition 5.12: Extended projection is *precise* generalization of standard projection with respect to unnesting.

Proof: We need to show that $\pi_{X'}(\mu^*(r)) = \mu^*(\pi_X^e(r))$, where X' are all of the attributes of the completely unnested scheme X . We show inclusion both ways to prove the equivalence.

\subseteq Let t be a tuple in $\pi_{X'}(\mu^*(r))$. Then, t is the projection onto X' of some tuple which unnested from a tuple t_r in r . For $\pi_X^e(r)$, t_r will be projected onto X and possibly combined with other tuples in an extended union. In any case, when unnested, the tuple t will be in the result.

\supseteq Let T be a set of tuples in $\mu^*(\pi_X^e(r))$ such that all tuples in T unnested from a single tuple t in $\pi_X^e(r)$. Then, there are two cases: either t came directly from a projection of r , or t is a combination of tuples in the projection of r .

Case 1: Suppose t came directly a tuple in a projection of r . Then, the projection hasn't been altered by the extended union, and since unnest commutes with projection [FT], all tuples in T will be in $\pi_{X'}(\mu^*(r))$.

Case 2: Suppose t is the extended union of two or more tuples in the projection of r . Then all of these tuples will be combined only where they agree on non-leaf attributes of the scheme tree for $\pi_X(r)$. Now,

the unnest of r will not eliminate any of these tuples, so the projection onto X' will return all tuples in T . \square

5.3 Closure of PNF Relations Under the Extended Operators

Theorem 5-3. *The class of PNF relations is closed under extended union, extended intersection, extended difference, cartesian product, extended natural join, extended projection, and selection.*

Proof: The proofs for each operator are presented below.

•**Extended Union**— We show that for any relation structures $\mathcal{R} = \langle R, r_1 \rangle$ and $S = \langle R, r_2 \rangle$ with attribute set E_R that $\mathcal{T} = \mathcal{R} \cup^e S$ is a PNF relation.

By definition of \cup^e , \mathcal{T} has scheme R with attribute set E_R . Let the instance of \mathcal{T} be r_3 . We need to show that, in r_3 , $A \rightarrow E_R$, where A is the set of zero order attributes of E_R . Suppose it does not. Then two tuples t_1 and t_2 in r_3 must agree on A and yet disagree on E_R . Now t_1 (and likewise t_2) either was carried over in total from r_1 or r_2 since it disagreed on A with all tuples in the other relation, or was created from tuples, one each in r_1 and r_2 which agreed on A and had the values of their higher order attributes combined with a recursive application of extended union. Thus there are four cases:

Case 1. t_1 and t_2 both carried over in total:

t_1 and t_2 cannot both come from one relation as each is in PNF and if t_1 agrees with t_2 on A then they agree on E_R . They cannot come from different relations as they agree on A and yet each is required by the definition of extended union to disagree with all other tuples in the other relation on A . Thus we have a contradiction for case 1.

Case 2. t_1 carried over in total and t_2 created from a tuple in each of r_1 and r_2 :

Suppose t_1 came from r_1 . Then t_1 disagrees with all tuples in r_2 on A . But t_2 was created from tuples that agreed on A , one in each of r_1 and r_2 . The argument for t_1 coming from r_2 is symmetric, and so case 2 leads to a contradiction.

Case 3. Symmetric to case 2 with t_1 and t_2 interchanged.

Case 4. t_1 and t_2 both created from a tuple in each of r_1 and r_2 :

Since t_1 and t_2 agree on A then all tuples in r_1 and r_2 from which they were created agree on A . Thus all tuples from r_1 must be the same tuple as $A \rightarrow E_R$ holds in r_1 . The symmetric argument holds for r_2 . Thus t_1 and t_2 were both created from the same two tuples, by an identical operation, and, therefore, agree on E_R . Thus we have a contradiction for case 4.

Since cases 1-4 all produced a contradiction the hypothesis is false and indeed $A \rightarrow E_R$ in r_3 and so \mathcal{T} is in PNF.

•**Extended Intersection**— This proof is the same as for extended union except that there is only one case in the case analysis that applies, case 4.

•**Extended Difference**— This proof is the same as for extended union except we need only consider tuples carried over in total from just r_1 .

•**Cartesian Product**— Let $\mathcal{V} = \langle V, v \rangle = \mathcal{R} \times \mathcal{S}$, where $\mathcal{R} = \langle R, r \rangle$ and $\mathcal{S} = \langle S, s \rangle$. We assume that the attributes have been renamed so that $E_R \cap E_S = \emptyset$. Then $E_V = E_R \cup E_S$. We show that $AB \rightarrow E_R E_S$ holds in v where A is the set of zero order attributes in E_R and B is the set of zero order attributes in E_S . Suppose it does not. Then two tuples t_1 and t_2 in v must agree on AB and yet disagree on $E_R E_S$. Assume the disagreement is in E_R as a symmetric argument can be made for E_S .

We have $A \rightarrow E_R$ in r since \mathcal{R} is in PNF. We also have that each tuple in v agrees with some tuple in r on E_R . Thus there are tuples in r that agree with t_1 and t_2 on E_R . Since t_1 and t_2 agree on AB they agree on A , but, as assumed, disagree on $E_R E_S$ and so disagree on E_R . Thus, $A \rightarrow E_R$ does not hold in r which is a contradiction. Therefore, the hypothesis is false and \mathcal{V} is in PNF.

•**Extended Natural Join**— Let $\mathcal{V} = \langle V, v \rangle = \mathcal{R} \bowtie^e S$, where $\mathcal{R} = \langle R, r \rangle$ and $S = \langle S, s \rangle$. We have that $E_V = E_R E_S$. Let $X = E_R \cap E_S$, $A = E_R - X$ and $B = E_S - X$. Let $A_z A_h = A$, where A_z are the zero order attributes of A and A_h are the higher order attributes of A . Similarly, let $B_z B_h = B$ and $X_z X_h = X$.

We show that $A_z B_z X_z \rightarrow E_R E_S$ holds in v . Suppose it does not. Then two tuples t_1 and t_2 in v must agree on $A_z B_z X_z$ and yet disagree on $E_R E_S$. This disagreement is either on A_h , B_h , or X_h . If the disagreement is on A_h or B_h then the arguments of cartesian product apply and a contradiction is reached. If the disagreement is on X_h then the argument of case 4 of union applies since the tuples from which t_1 and t_2 came must be identical in r and s as FDs $A_z X_z \rightarrow E_R$ holds in r and $B_z X_z \rightarrow E_S$ holds in s . Thus we reach a contradiction and so \mathcal{V} is in PNF.

•**Extended Projection**— When an extended projection operation is applied to a relation we do not change any FDs that hold in the nested relations of each tuple, as we either take the nested relation in total or eliminate it. Also if all nested relations meet the requirements to be in PNF then a single tuple containing these nested relations is automatically in PNF. Therefore, we can apply the proof for union since extended projection is a tupewise extended union of the tuples resulting from a normal projection operation, each of which we determined was a PNF relation.

•**Selection**— A subset of the tuples of a relation cannot violate an FD that holds on the entire relation, so any selection of tuples from a PNF relation produces a PNF relation. \square

Chapter 6

Equivalence of the Relational Calculus and the Relational Algebra

In this chapter, we prove that the relational calculus and algebra as extended to handle nested relations are equivalent. We first show that all relational algebra expressions can be expressed in the safe relational calculus, and then the inverse relationship.

6.1 Reduction of Relational Algebra to Relational Calculus

Theorem 6-1. *If E is a relational algebra expression, then there is a safe expression in the relational calculus equivalent to E .*

Proof: The proof is by induction on the number of occurrences of operators in E . The basis and the five cases (Cases 1-5) for \cup , $-$, \times , π , and σ are as in [UII]. We need two more cases for the operators ν and μ .

Case 6: $E = \nu_{B=(A_1A_2\dots A_k)}(E_1)$. Let E_1 be equivalent to safe expression $\{t^{(n)} \mid \psi_1(t)\}$ and let attribute A_i correspond to the j_i 'th attribute, for $1 \leq i \leq k$, and let all attributes not among the A_i correspond to the j_l 'th attribute, for

$k < \ell \leq n$. Then E is equivalent to

$$\{t^{(n-k+1)} | (\exists u)(\psi_1(u) \wedge \bigwedge_{m,\ell} t[m]=u[j_\ell] \\ \wedge t[j_1]=\{w^{(k)} | (\exists v)(\psi_1(v) \wedge \bigwedge_{m,\ell} t[m]=v[j_\ell] \wedge \bigwedge_{i=1}^k w[i]=v[j_i])\})\})\}$$

where m ranges over $[1 : j_1 - 1, j_1 + 1 : n - k + 1]$ as ℓ ranges over $[k + 1 : n]$.

Since sets-of-values are being created, we need to check if the elements are from a finite domain, and, in this case, they are from $DOM(\psi_1)$, so this expression is safe.

Case 7: $E = \mu_A(E_1)$. Let E_1 be equivalent to safe expression $\{t^{(n)} | \psi_1(t)\}$ and let attribute A correspond to the i th attribute and let the arity of A be k .

Then E is equivalent to

$$\{t^{(n+k-1)} | (\exists u)(\psi_1(u) \wedge \bigwedge_{m,\ell} t[m]=u[\ell] \wedge (\exists w)(w \in u[i] \wedge \bigwedge_{p,q} t[p]=w[q]))\}$$

where m ranges over $[1 : i - 1, i + k : n + k - 1]$ as ℓ ranges over $[1 : i - 1, i + 1 : n]$, and where p ranges over $[i : i + k - 1]$ as q ranges over $[1 : k]$.

As in case 6, the elements of $DOM(\psi_1)$ are the only ones used in this expression, so it is safe as well. \square

6.2 Reduction of Relational Calculus to Relational Algebra

Theorem 6-2. *If E is a safe expression in the relational calculus then there*

is a relational algebra expression equivalent to E .

In order to prove the theorem we must first establish some basic results.

Lemma 6-1. *If ψ is any formula in tuple calculus then there is an equivalent formula ψ' of tuple calculus with no occurrences of \wedge or \vee . If ψ is safe, so is ψ' .*

Proof: See [Ull], Lemma 5-2. □

Lemma 6-2. *If ψ is any formula in tuple calculus then there is an algebra expression for $DOM(\psi)$.*

Proof: Completely unnest each relation and constant that contains nested relations, and appears in ψ . Then, as in [Ull], use projection and union to form a unary relation, containing all possible values that are mentioned in ψ . □

Our proof of the theorem mirrors the proof in [Ull] of the equivalence of the (1NF) relational calculus and algebra. In [Ull], an algebra expression was created which produced a unary relation E of all values either mentioned explicitly as constants in the calculus expression or exists in any relation mentioned in the calculus expression. Each atom of the calculus expression is then translated as a function of $\times_{i=1}^n E$ where n is the number of attributes in all tuples variables being used in the subexpression where the atom occurs. The

relation E is basically a *domain* of values from which the calculus expression must create the tuples in the result. However, when we move to $\neg 1NF$ relations, it is not possible to create a domain of values using this technique. Each tuple variable may range over values that are nested relations, and so to include all possible nested relations, we would have to have a technique for creating a powerset using the relational algebra. Since it is not possible to create a powerset using the algebra (see Appendix A), we will use subsets of all possible tuples for each tuple variable and each component of a tuple variable that is defined as a nested relation. These *limited domains*, when completely unnested, contain all possible tuples from which the calculus expression will select tuples for a completely unnested result.

Definition 6.1: A *limited domain* for a tuple variable t , denoted D_t , appearing in a safe calculus expression, $\{x \mid \psi(x)\}$, is an extended relational algebra expression which produces a $\neg 1NF$ relation r which, when completely unnested, contains all tuples, made up of values from $DOM(\psi)$, which need to be tested for inclusion or exclusion, by the atoms of the calculus expression referring to t . The $\neg 1NF$ tuples which actually are tested by t will be an *extended* intersection of D_t .

If there is a subformula of the form $(\exists t)(p(t))$, then a limited domain for t contains tuples to be *included* in the result if they satisfy p . If there is a subformula of the form $\neg(\exists t)(p(t))$, or $(\forall t)(\neg p(t))$, then a limited domain for t contains tuples to be *excluded* from the result if they satisfy p . In the main

body of the proof we present a way to construct an algebra expression which performs the proper inclusions and exclusions on the tuples in each limited domain. We use the extended operators defined in Chapter 5 to include and exclude tuples from nested relations.

Lemma 6-3. *Given a safe tuple calculus expression $\{t \mid \psi(t)\}$, there is an algebra expression D_{t_i} for a limited domain of each tuple variable t_i mentioned in ψ , or any nested expression of ψ .*

Proof: Since the calculus expression is safe, we claim that we can determine each D_{t_i} by scanning the expression for named relations and constants. Each atom in the expression constrains the values that a tuple variable or a component of a tuple variable may assume.

The following algorithm examines each atom in the expression and adds algebra expressions to each domain so that the possible values which that atom references will be included in the domain. The intuition behind this algorithm is as follows. When atoms refer to named relations and constants the reference is direct and known. However, when the atoms refer only to tuple variables, then the reference is indirect, and must be solved in terms of tuple variables which have direct and known references. In addition, there may be more than one atom which references a particular attribute of a tuple variable, and so we may get multiple expressions for each domain. Thus, as the algorithm creates the algebra expression for each domain, it also creates a graph which

tells us how to solve the indirect references in our algebra expressions. Let D_i^t be the algebra expression for the limited domain of the i th attribute of tuple variable t . The graph will be constructed of nodes, directed edges, and directed and-edges. A directed and-edge is a single edge which goes from a single node to a set of one or more nodes. Nodes will be labeled with the limited domain variable, and edges will be labeled with algebra expressions which may become part of the limited domain of the node from which the edges emanate, and a special label if the atom for which the label was created involved a $>$ comparison. Atoms involving $>$ comparisons usually do not add anything to the limited domains that would not be included by another type of atom. However, there is the special case where two atoms define a range of values, which is the only specification of the limited domain of some component of a tuple variable; e.g., $x[1] > 2 \wedge \neg(x[1] > 5)$. In this case, we use the algebra expression for $DOM(\psi)$ (Lemma 6-2), so that we get every value in the range. Note that if there are values in the range that are not in $DOM(\psi)$ then the expression is not safe.

Algorithm 1

```

Create a graph with one node labeled  $RC$ , standing for named relations
and constants;
For each tuple variable  $t$ 
do
    let  $k$  denote the arity of  $t$ ;
    create  $k$  nodes in the graph, labeled  $D_i^t$ ,  $1 \leq i \leq k$ 
end do
For each atom in the calculus expression
do

```

case atom

$t \in r$:

let k denote the arity of t ;

add directed edges from D_i^i to RC , $1 \leq i \leq k$;

for $1 \leq i \leq k$, label the edge from D_i^i to RC with $\pi_i(r)$;

$t \in u[j]$:

let k denote the arity of t ;

add directed edges from D_i^i to D_u^j , $1 \leq i \leq k$;

for $1 \leq i \leq k$, label the edge from D_i^i to D_u^j with $\pi_i(\mu_1(D_u^j))$;

$t[j] \theta a$ or $a \theta t[j]$, $\theta \in \{=, >\}$:

add a directed edge from D_i^i to RC ;

label the edge C , where C is a new unary relation

containing the single tuple $\langle a \rangle$;

add a special label, Φ , to the edge if $\theta = >$;

$t[j] \theta u[\ell]$, $\theta \in \{=, >\}$:

add directed edges from D_i^j to D_u^ℓ and from D_u^ℓ to D_i^j ;

label the edge from D_i^j to D_u^ℓ with D_u^ℓ ;

label the edge from D_u^ℓ to D_i^j with D_i^j ;

add a special label, Φ , to each edge if $\theta = >$;

$t[j] = \{u^{(\ell)} \mid \psi'(u)\}$:

add a directed and-edge from D_i^j to the set of nodes D_u^i ,

$1 \leq i \leq \ell$;

label the and-edge $\nu_{1=(1,2,\dots,\ell)}(D_u^1 \times D_u^2 \times \dots \times D_u^\ell)$;

end case

end do

Mark node RC ;

Let \mathcal{D} be the algebra expression for $DOM(\psi)$;

While some node in the graph is not marked

do

Choose an unmarked node N with at least one edge, without a special label Φ , directed towards a marked node, or at least one and-edge directed towards a set of nodes, all of which are marked, or

if neither of the above cases applies, at least one edge, with a special label Φ , directed towards a marked node;

If the special case using label Φ was invoked then let $C = \mathcal{D}$

else let $C = \emptyset$;

Set the algebra expression for the domain labeled N to

$L_1 \cup L_2 \cup \dots \cup L_p \cup C$, where p is the number of edges

and and-edges directed from N to marked nodes,
 and L_i is the label of the i th such edge;
 Mark node N
 end do
 For each tuple variable t with arity k , set D_t to $D_t^1 \times D_t^2 \times \dots \times D_t^k$.

The correctness of this algorithm follows from the following arguments. First, we show that the algorithm halts. Suppose that it does not halt. Then there must be unmarked nodes in the graph and no path from them to the node RC . Consider the tuple variables naming these nodes. The variables are used only in atoms which never refer to any of the relations or constants in the expression. So they can take unknown values and still satisfy the expression. As there is no way to determine these values, the expression must be unsafe. This is a contradiction, and so the algorithm must halt.

The expressions are correct if each limited domain includes all possible tuples which the calculus expression will include or exclude from the result. Suppose some limited domain D_t does not include all such tuples. Then, there must be an atom in which t appears that must test values not appearing in D_t . The atom cannot compare t or a component of t to a named relation or constant using $=$ or \in , since these tuples always included due to the initial marking of node RC . If the comparison involves $>$, then there must be other comparisons involving t in order for the expression to be safe. Thus, the atom compares t , or a component of t , with either the component of another tuple variable, x , or a set of tuples, u , created by a nested calculus expression.

Let us assume that the entire tuple variable is being accessed, otherwise add the appropriate superscript to the limited domain variable if only a component is being accessed.

In the first case, either D_t , D_x , or both D_t and D_x , are determined by other comparisons in other atoms. Consider each of these subcases. (1) If D_t is determined by comparisons within other atoms and D_x is not, then the comparison involving t and x does not add any tuples, and D_x is a subset of D_t . (2) If D_x is determined by comparisons within other atoms and D_t is not, then D_t is a subset of D_x and we must make a new argument for D_x . If we continue to invoke this subcase, a trivial induction shows that we eventually run out of tuple variables and if the last variable used is not expressed in terms of named relations or constants then the expression is not safe. (3) If both D_x and D_t are determined by other comparisons then the algorithm either adds D_x to D_t or D_t to D_x , and so subcase 1 and subcase 2 apply, respectively.

In the case of comparison with a set of tuples u , it must be that the limited domain D_u does not contain all possible tuples, and so we make a new argument for D_u . This case can only be invoked as long as there are still nested calculus expressions. Once we have exhausted them, the first case applies.

Thus, either the expression is unsafe, or we have included all the necessary tuples in our limited domains, and so the algorithm is correct. \square

Proof of Theorem 6-2: Let $\{t \mid \psi(t)\}$ be a safe tuple calculus expression. We construct an equivalent algebra expression. By Lemma 6-3 we have an algebra expression D_x for each tuple variable x mentioned in ψ . By Lemma 6-1 we may assume that ψ has only the operators \vee , \neg , and \exists .

We prove by induction on the number of operators in a subformula ω of ψ that if ω has free variable s , then

$$D_s \cap^e \{s \mid \omega(s)\}$$

has an equivalent expression in relational algebra. Then, as a special case, when ω is ψ itself, we have an algebraic expression for

$$D_t \cap^e \{t \mid \psi(t)\}$$

Since ψ is safe, intersection with D_t does not change the relation denoted, so we shall have proved the theorem. We use the extended intersection operator since D_t may contain nested relations which need to be intersected with the corresponding nested relations produced by ψ .

In order to avoid problems where $\nu_A(\mu_A(r)) \neq r$, and so that the extended operators do not interact improperly, we assume each database relation (r, q, \dots) , their nested relations, and relations created by collecting constants into a limited domain, have an implicit keying attribute (or set of attributes) whose value uniquely determines the values of all other attributes. We consider this attribute to be added to each relation before it is used and removed

when the relation is projected or presented as the final result, using appropriate algebra operations. A key can always be added to a relation by making a side-by-side copy of the relation with itself and using one of the copies as a key. If a nested relation needs a key then after ensuring the relations in which the nested relation resides are keyed, we unnest that nested relation, make a side-by-side copy to gain a key and then re-nest adding the key to the nested relation. If r is a relation with arity n , then a side-by-side copy can be made as follows:

$$\sigma_{1=n+1 \wedge 2=n+2 \wedge \dots \wedge n=n+n}(r \times r)$$

The first n attributes of this new relation then serve as the key. Fewer attributes can be used if they are a primary key for relation r , and the above expression can be projected to retain only those attributes in the key portion. Note that relations which are in partitioned normal form already satisfy these key constraints.

We now proceed with the inductive proof.

Basis: Zero operators in ω . Then ω is an atom, which we may take to be in one of the forms described in Chapter 4. In order to specify an algebra expression for these atoms, which may, as themselves, specify infinite relations, we need to operate on an expression $D = D_{s_1} \times D_{s_2} \times \dots \times D_{s_n}$, where the s_i are *all* free tuple variables of the formula ω of which this atom is currently a part.

The atoms are thus translated:

1. $s \in r$: Replace D_s in D by r .
2. $s \in t[i]$: Let p_1, p_2, \dots, p_k be the attributes of D_s in D , let q^* be the i th attribute of D_t in D , and let q_1, q_2, \dots, q_k be the attributes of q^* .
Let D' be

$$\sigma_{p_1=q_1 \wedge \dots \wedge p_k=q_k}(\mu_{q^*}(D))$$

Then the desired expression is

$$\pi_X(\sigma_F(D \times D'))$$

where X is the attributes of D and F is a predicate which matches all attributes of D except q^* with the corresponding attributes in D' . By unnesting we can access the elements of $t[i]$ using standard relational algebra operators. In D' the selection picks out those values corresponding to tuple variable s 's domain D_s . This gives us a set of values which we can use to choose the tuples of D which have the sets in $t[i]$ of which s is a member. The final expression gives this result.

3. $a \theta s[i], s[i] \theta a, s[i] \theta t[j]$: Let p be the i th attribute of D_s and q be the j th attribute of D_t , then desired algebra expressions are, respectively:

$$\sigma_{a \theta p}(D) \quad \sigma_{p \theta a}(D) \quad \sigma_{p \theta q}(D)$$

4. $s[i] = \{u^{(j)} \mid \psi'(u, t_1, t_2, \dots, t_n)\}$: We have s as one of t_1, t_2, \dots, t_n , and j as the arity of a new tuple variable u . Let E' be an algebra expression

for ψ' and k be the arity of D . The desired algebra expression is D with D_i^j replaced by

$$\pi_{k+1}(\nu_{k+1=(k+1,k+2,\dots,k+j)}(E')).$$

Since E' is an expression on $D \times D_u$, the $k+1$ through $k+j$ attributes of E' will be tuples corresponding to u . Since this is a nested expression we apply the nest operation and use this new expression in place of D_i^j in the expression for this atom.

Induction: Assume ω has at least one operator and that the inductive hypothesis is true for all subformulas of ψ having fewer operators than ω . We now proceed to a case analysis covering each of the three operators. Let $D = D_{t_1} \times D_{t_2} \times \dots \times D_{t_n}$.

Case 1: $\omega(t_1, t_2, \dots, t_n) = \omega_1(t_1, t_2, \dots, t_n) \vee \omega_2(t_1, t_2, \dots, t_n)$ where the t_i are the free tuple variables in the expression ω . We do not require ω_1 or ω_2 to use any or all of the t_i . Let E_1 be an algebraic expression for

$$D \cap^e \{t_1, t_2, \dots, t_n \mid \omega_1(t_1, t_2, \dots, t_n)\}$$

and E_2 an algebraic expression for

$$D \cap^e \{t_1, t_2, \dots, t_n \mid \omega_2(t_1, t_2, \dots, t_n)\}.$$

Then the desired expression is

$$E_1 \cup^e E_2.$$

Recall that, in Chapter 5, we outlined a procedure for expressing extended union in terms of the basic relational algebra operators.

Case 2: $\omega(t_1, t_2, \dots, t_n) = \neg\omega_1(t_1, t_2, \dots, t_n)$. Let E_1 be an algebraic expression for

$$D \cap^e \{t_1, t_2, \dots, t_n \mid \omega_1(t_1, t_2, \dots, t_n)\}$$

then

$$D -^e E_1$$

is an expression for

$$D -^e \{t_1, t_2, \dots, t_n \mid \omega_1(t_1, t_2, \dots, t_n)\}$$

which is equivalent to

$$D \cap^e \{t_1, t_2, \dots, t_n \mid \neg\omega_1(t_1, t_2, \dots, t_n)\}.$$

As for case 1, refer to Chapter 5 to see that extended difference is expressible in terms of the basic relational algebra operators.

Case 3: $\omega(t_1, t_2, \dots, t_n) = (\exists t_{n+1})(\omega_1(t_1, t_2, \dots, t_{n+1}))$. Let E_1 be an algebraic expression for

$$D \times D_{t_{n+1}} \cap^e \{t_1, t_2, \dots, t_{n+1} \mid \omega_1(t_1, t_2, \dots, t_{n+1})\}$$

Since ψ is safe ω is safe. The expression $\omega_1(t_1, t_2, \dots, t_{n+1})$ is never true unless t_{n+1} is in the set $DOM(\omega)$, which is a subset of $DOM(\psi)$. Therefore $\pi_J(E_1)$, $J =$ the attributes of t_1, t_2, \dots, t_n , denotes the relation

$$D \cap^e \{t_1, t_2, \dots, t_n \mid (\exists t_{n+1})(\omega_1(t_1, t_2, \dots, t_{n+1}))\}$$

which completes the induction, and proves the theorem. \square

6.3 Examples

To illustrate Lemma 6-2, consider the following calculus expression:

$$\{t^{(2)} \mid (\exists s)((s \in r \vee s \in q) \wedge s[1] = t[1] \wedge t[2] = \{u^{(2)} \mid u \in s[2] \vee u \in s[3]\})\}$$

Assume that r and q are relations with three attributes, the second and third attributes being nested relations having two attributes each.

Before the marking phase of the algorithm the graph is as shown in Figure 6-1. During the marking phase, RC is marked. Then D_s^1 , D_s^2 , and D_s^3 are marked and the term D_t^1 is not included in the expression for D_s^1 , since D_t^1 is not yet marked. Then, D_t^1 , D_u^1 , and D_u^2 can be marked, and, finally, D_t^2 is marked since all nodes at the end of the and-edge are marked. The algebra expressions at the end of the marking phase are:

$$D_s^1 = \pi_1(r) \cup \pi_1(q)$$

$$D_s^2 = \pi_2(r) \cup \pi_2(q)$$

$$D_s^3 = \pi_3(r) \cup \pi_3(q)$$

$$D_t^1 = D_s^1$$

$$D_t^2 = \nu_{1=(1,2)}(D_u^1 \times D_u^2)$$

$$D_u^1 = \pi_1(\mu_1(D_s^2)) \cup \pi_1(\mu_1(D_s^3))$$

$$D_u^2 = \pi_2(\mu_1(D_s^2)) \cup \pi_2(\mu_1(D_s^3))$$

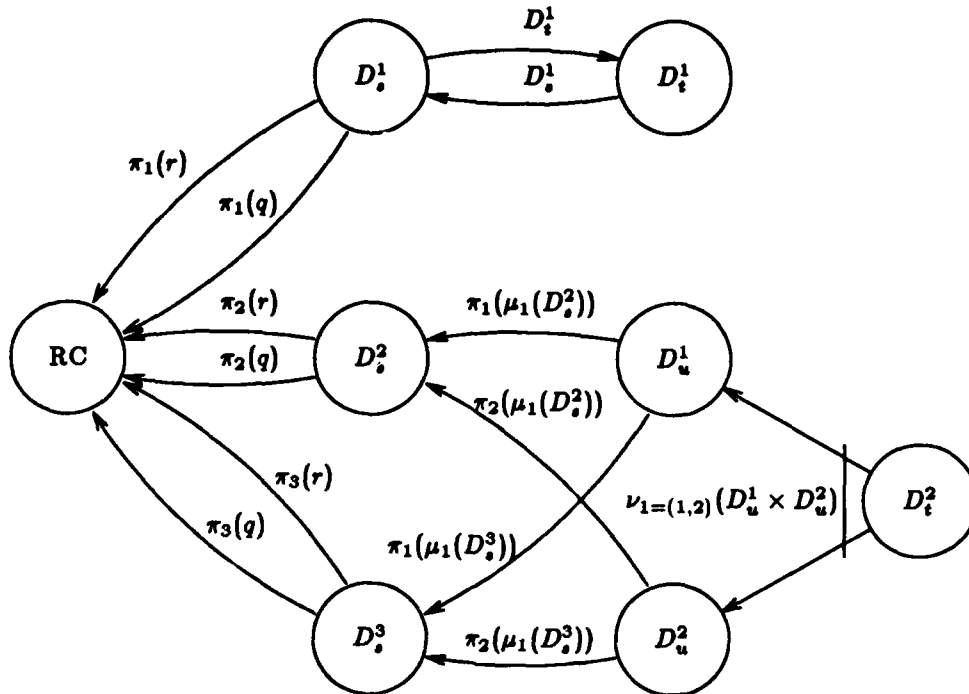


Figure 6-1. Graph produced by Algorithm 1.

Substituting for the variables and applying the final cartesian products, we have:

$$D_s = (\pi_1(r) \cup \pi_1(q)) \times (\pi_2(r) \cup \pi_2(q)) \times (\pi_3(r) \cup \pi_3(q))$$

$$D_t = (\pi_1(r) \cup \pi_1(q)) \times \nu_{1=(1,2)}((\pi_1(\mu_1(\pi_2(r) \cup \pi_2(q))) \cup \pi_1(\mu_1(\pi_3(r) \cup \pi_3(q)))) \\ \times (\pi_2(\mu_1(\pi_2(r) \cup \pi_2(q))) \cup \pi_2(\mu_1(\pi_3(r) \cup \pi_3(q))))))$$

$$D_u = (\pi_1(\mu_1(\pi_2(r) \cup \pi_2(q))) \cup \pi_1(\mu_1(\pi_3(r) \cup \pi_3(q)))) \\ \times (\pi_2(\mu_1(\pi_2(r) \cup \pi_2(q))) \cup \pi_2(\mu_1(\pi_3(r) \cup \pi_3(q))))$$

For a complete example of the transformation process of Theorem 6-2,

consider the following calculus expression:

$$\{t^{(2)} | (\exists s)(s \in r \wedge t[1]=s[1] \wedge t[2]=\{u^{(2)} \mid u \in s[2] \wedge u[2] \leq '1970'\})\}$$

where r is a relation on $R=(\text{course}, \text{Date})$, $\text{Date}=(\text{month}, \text{year})$. This query is asking for all courses and the set of dates for the course with a year at most 1970.

Using the methodology of section 6.2 we translate this TRC expression into an equivalent relational algebra expression. We start by transforming the expression so that \neg , \vee , and \exists are the only operators present.

$$\{t^{(2)} | (\exists s)(\neg(\neg(s \in r) \vee \neg(t[1]=s[1]) \vee \neg(t[2]=\{u^{(2)} \mid \neg(\neg(u \in s[2]) \vee (u[2] > '1970'))\})))\}$$

The domains corresponding to each tuple variable are

$$D_s = \pi_1(r) \times \pi_2(r),$$

$$D_t = \pi_1(r) \times \nu_{1=(1,2)}(\pi_1(\mu_1(\pi_2(r))) \times (\pi_2(\mu_1(\pi_2(r))) \cup \{(1970)\})),$$

$$D_u = \pi_1(\mu_1(\pi_2(r))) \times (\pi_2(\mu_1(\pi_2(r))) \cup \{(1970)\}).$$

We now proceed with the translation. Translate each atom:

$$s \in r \quad \rightarrow \quad E_1 = D_t \times r$$

$$t[1]=s[1] \quad \rightarrow \quad E_2 = \sigma_{1=3}(D_t \times D_s)$$

$$t[2]=\{\dots\} \quad \rightarrow \quad E_3 = (\pi_1(r) \times \pi_5(\nu_{5=(5,6)}(E'))) \times D_s$$

where E' is the algebra expression for $\{\dots\}$.

Translate negation and disjunction:

$$E_4 = (D_t \times D_s) -^e (((D_t \times D_s) -^e E_1) \cup^e ((D_t \times D_s) -^e E_2) \cup^e ((D_t \times D_s) -^e E_3))$$

Translate existential quantifier and the final expression is:

$$E = \pi_{1,2}(E_4)$$

E' is determined similarly.

Translate the atoms:

$$u \in s[2] \quad \rightarrow \quad E_1'' = \sigma_{4=6 \wedge 5=7}(\mu_4(D_t \times D_s \times D_u))$$

$$E_1' = \pi_{1,2,3,4}(\sigma_{1=5 \wedge 2=6 \wedge 3=7}((D_t \times D_s \times D_u) \times E_1''))$$

$$u[2] > '1970' \quad \rightarrow \quad E_2' = \sigma_{6 > '1970'}(D_t \times D_s \times D_u)$$

Translate negation and disjunction (and since there are no existential quantifiers) giving the result:

$$E' = (D_t \times D_s \times D_u) -^e (((D_t \times D_s \times D_u) -^e E_1') \cup^e E_2')$$

This ends the translation process. For comparison purposes the query as it would directly be written in the algebra is

$$\nu_{Date}(\sigma_{year \leq '1970'}(\mu_{Date}(r)))$$

This assumes that the course values are all unique in r . If not we would need to add a key to the relation so that the nest does not combine sets that were separate in the beginning.

Chapter 7

Null Values in \neg 1NF Relational Databases

A problem may arise in a \neg 1NF representation of a database. Consider a database of employees, their children and their skills. Figure 7-1 shows an example 1NF version of this database, and Figure 7-2 shows the corresponding \neg 1NF version. If we have an employee with several skills and no children, then, in the database of Figure 7-1, we simply add tuples to the (employee, skill) relation and add nothing to the (employee, child) relation. Now, consider the representation of this information in the \neg 1NF relation of Figure 7-2. In this relation, a tuple seemingly requires that employees have at least one skill and at least one child before they can be entered into the database. The solution is to employ empty sets. This is the same problem encountered by users of a universal relation system $[K+]$. In the \neg 1NF case, null values can occur in nested relations as well as for nondecomposable attributes. The empty set is, in effect, a type of null value.

The various nulls which have been proposed vary in the type of incomplete information they represent or the degree of the incompleteness. For example, we may have different nulls to represent both the non-existence of a value and the existence of a value that is not precisely known. In this chapter we make the open world assumption. That is, we assume that just because a tuple is not in a relation does not mean it should not be there. The best we can

employee	child	employee	skill
Smith	Sam	Smith	typing
Smith	Sue	Smith	filig
Jones	Joe	Jones	typing
Jones	Mike	Jones	dictation
		Jones	data entry

Figure 7-1. 1NF representation of employee relation.

employee	Children	Skills
	child	skill
Smith	Sam	typing
	Sue	filig
Jones	Joe	typing
	Mike	dictation
		data entry

Figure 7-2. Employee relation in γ 1NF.

do at any point in time is enter tuples into a relation that we know currently belong there. In addition, if we know partial information about a tuple then the unknown information is represented using null values.

A different, although compatible, source of nulls occurs when we attempt to represent multiple relationships among data in a single relation (an extreme example being the *universal relation assumption* [FMU]). For example, in a single relation we may want to represent facts about suppliers, parts, and associations stating which suppliers supply which parts. If a supplier is currently not supplying a part, then the part attributes of the relation must contain null values. If null values are not allowed, then a non-supplying supplier

could not be represented in this relation.

Thus, the same motivation which requires us to add null values to a traditional 1NF database holds for \neg 1NF databases. However, the need for nulls is even more critical in a \neg 1NF database since otherwise we lose some of its advantages. Since we have the ability to represent multiple relationships in a single \neg 1NF relation without the problems of redundancy that doing so in a 1NF relation would entail, we must also deal with the fact that one or more of those relationships may be unknown or non-existent at some time.

The remainder of this chapter is organized as follows. In section 7.1, we summarize a formal treatment of null values in the traditional relational model. The no-information, unknown, and nonexistent interpretation of nulls are discussed. We show that reasonable extensions to the traditional relational operators are possible under the open world assumption. These extensions serve as a basis for the main results of this chapter, the extension to \neg 1NF. In section 7.2, we extend the null value theory presented in section 7.1 to \neg 1NF relations, and further extend the operators of Chapter 5 to deal with null values. Finally, in section 7.3, we discuss dependency theory, shedding some new light on the problem of nulls when dealing with functional and multivalued dependencies, and their axiomatization.

7.1 Null Values in 1NF Relations

In this section, we briefly review the basic concepts that concern null values in 1NF relations. The presentation is based on some of the work of Zaniolo [Zan1, Zan2]. We distinguish between three types of nulls:

- **ni** – no-information,
- **unk** – unknown, and
- **dne** – nonexistent (or does not exist),

and extend each domain to include these null values.

Previous approaches have usually assumed only one of the interpretations is valid, *unknown* by [Bis1, Cod2, Gran, Mai1], and *nonexistent* by [Lie2, Lie3, Sci, Zan1]. In [Vas2] a combination of the two is proposed in which nonexistence is considered an inconsistent state of data. Finally, Zaniolo [Zan2] provides a unified approach to nulls with the use of a *no-information* null. This null is less informative than either an *unknown* or a *nonexistent* null, and can be used to approximate both when we don't know whether or not a value exists. As this is the most complete and conceptually sound approach proposed to date, it forms the basis of our extensions to \neg 1NF relations.

Other proposals for nulls are rather sophisticated, involving partial specification [Lip1, Lip2, IL1, IL2], probability distributions [Won], and con-

ditional tuples [KW], but it could be argued "that the complexity of their management is not justified by their richer semantics" [AM; 233].

7.1.1 Basic Concepts

When dealing with incomplete information, we talk about a strength ordering of information in which certain tuples will be *more informative* than others, say by having a previously unknown value replaced by an actual value, or by finding out that a value for which we previously had no-information is now known not to exist. In order to compare values for this purpose we define a *greatest lower bound* function which tells us the most information we can infer from two values from the same extended domain.

Definition 7.1: Let $\{d_1, d_2, \dots, d_n\}$ be a domain and $D = \{d_1, d_2, \dots, d_n, \text{unk}, \text{dne}, \text{ni}\}$ the corresponding extended domain. A *greatest lower bound* function, $glb(a, b)$, between two values a and b from D is defined in Figure 7-3.

$b \backslash a$	d_1	d_2	\dots	d_n	unk	dne	ni
d_1	d_1	unk	unk	unk	unk	ni	ni
d_2	unk	d_2	unk	unk	unk	ni	ni
\vdots	unk	unk	\ddots	unk	unk	ni	ni
d_n	unk	unk	unk	d_n	unk	ni	ni
unk	unk	unk	unk	unk	unk	ni	ni
dne	ni	ni	ni	ni	ni	dne	ni
ni	ni	ni	ni	ni	ni	ni	ni

Figure 7-3. Definition of glb function.

This information can also be represented as a lattice with ni as the bottom

element, **unk** and **dne** as more informative nulls than **ni**, and actual values d_1, d_2, \dots, d_n as more informative than **unk**. See Figure 7-4.

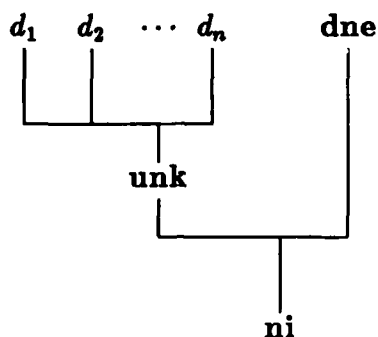


Figure 7-4. Information lattice.

Note that the **dne** null is special in that it does not have a possible, more informative, replacement. It is, in fact, a special “value” in itself, for which equality is meaningful. That is, $\mathbf{dne} = \mathbf{dne}$, but $\mathbf{ni} \neq \mathbf{ni}$ and $\mathbf{unk} \neq \mathbf{unk}$. Other restrictions on relations with **dne** nulls will be discussed in section 7.3.

We now define an information-wise strength ordering of tuples using the *glb* function as follows:

Definition 7.2: An X -value s is said to be *more informative* than a Y -value t , written $s \geq t$, if for each $B \in Y$, if $t[B]$ is not **ni** then $B \in X$, and for each $A \in X \cap Y$, $glb(t[A], s[A]) = t[A]$.

Conversely, if $s \geq t$ we say that t is *less informative* than s . The notion of *more informative* is synonymous to the concept of *subsumption*. We say s subsumes t when $s \geq t$. If we have two tuples in a relation such that one

is more informative than the other, then the less informative tuple is redundant and can be removed. Note that in the absence of nulls, this condition reduces to elimination of redundant identical tuples. If both $t \geq s$ and $s \geq t$, then we say t and s are *information-wise equivalent* and write $s \cong t$.

As a running example in this section, we use relation schemes $R_1 = (\text{employee}, \text{skill})$, and $R_2 = (\text{employee}, \text{child}, \text{skill})$.

Example 7.1: Let

$$t_1 = \langle \text{Smith}, \text{Bill}, \text{typing} \rangle, \quad t_2 = \langle \text{Smith}, \text{ni}, \text{unk} \rangle$$

denote E_{R_2} -values, and let

$$t_3 = \langle \text{Smith}, \text{unk} \rangle, \quad t_4 = \langle \text{Smith}, \text{typing} \rangle$$

denote E_{R_1} -values. Then, t_1 is more informative than t_2 , t_3 , and t_4 . Furthermore, $t_4 \geq t_2$, $t_4 \geq t_3$, and $t_2 \cong t_3$. \square

For certain relational operators it is convenient that all tuples be defined over the same set of attributes. With the availability of a *no-information* null we can extend tuples defined over different sets of attributes without changing the information content of the tuples. The extension is done by adding attributes used in one tuple and not in the other and assigning the value *ni* to these added attributes.

In order to find the most informative tuple which characterizes two other tuples we define the *meet* operator as follows:

Definition 7.3: The meet of an X -value, t_1 , and a Y -value, t_2 , is the XY -value, t , written, $t_1 \wedge t_2$, where for each attribute $A \in X \cap Y$, $t[A] = \text{glb}(t_1[A], t_2[A])$, and for each attribute $B \notin X \cap Y$, $t[B] = \text{ni}$.

Example 7.2: Using the tuples defined in Example 7.1 we find that

$$t_1 \wedge t_3 = t_2$$

$$t_1 \wedge t_4 = \langle \text{Smith}, \text{ni}, \text{typing} \rangle$$

□

We also generalize the notion of a tuple being an element, or a member of a relation as follows.

Definition 7.4: A tuple t is an x -element of a relation r , written $t \tilde{\in} r$, when there exists a tuple $s \in r$ such that $s \geq t$.

Thus an x -element of a relation is any tuple that is equal to or less informative than some tuple in the relation. We also write $t \tilde{\notin} r$ to denote $\neg(t \tilde{\in} r)$.

Given a set of tuples t_1, t_2, \dots, t_n , we can eliminate tuples in which all attributes have value ni (the *null tuple*) †, eliminate all tuples less informative than some other tuple, and extend all tuples by adding ni values for attributes not in the tuple but in some other tuple in the set. This is called *tuple set reduction* and is denoted by

$$\widehat{\{t_1, t_2, \dots, t_n\}}$$

† Even though a null tuple is subsumed by all tuples, it may be the only tuple in a relation, and thus should be eliminated.

The notion of being more informative can be extended to relations.

Definition 7.5: A relation r_1 is *more informative than*, or *subsumes*, a relation r_2 , written $r_1 \geq r_2$, when for each tuple $t_2 \in r_2$ there is a tuple $t_1 \in r_1$ with $t_1 \geq t_2$.

This \geq relationship is transitive and reflexive, leading to the following definition of *information-wise equivalence*.

Definition 7.6: The relations r_1 and r_2 are *information-wise equivalent*, written $r_1 \cong r_2$, when $r_1 \geq r_2$ and $r_2 \geq r_1$.

The equivalence relation \cong partitions the universe of relations into disjoint subclasses. Each class can be represented by a minimal relation in which no tuples in the relation are subsumed by a tuple in the same relation.

Definition 7.7: A relation r constitutes a *minimal representation* for a relation q when $r \subseteq q$, $r \cong q$, and $\nexists p \subset r$ such that $p \cong q$.

It is straightforward to show that the minimal representation of a relation is unique and therefore minimum.

7.1.2 Operators

In this section we briefly review extensions to the relational algebra operators to 1NF relations with nulls. We treat the dne null as any other domain value and, unless otherwise specified, any future reference to null will include only ni

and **unk** nulls. Some of this presentation is based on Section 12.4 of [Mai2].

Let $Rel\uparrow$ denote the sets of all relations having at least one null value and let Rel denote the set of all relations having no nulls, with $Rel\uparrow(R)$ and $Rel(R)$ denoting restrictions of $Rel\uparrow$ and Rel to relations on scheme R . We shall view a relation r in $Rel\uparrow(R)$ as representing a set of relations from $Rel(R)$ that subsume r . Each such relation in $Rel(R)$ is called a *possibility*. The set of possibilities for r is denoted by $POSS(r)$, which is defined as:

$$POSS(r) = \{q \mid q \in Rel(R) \text{ and } q \geq r\}$$

We extend the definition of relational operators to map sets of relations to other sets of relations. For sets P_1 and P_2 of relations and relational operator γ ,

$$\gamma(P_1) = \{\gamma(q) \mid q \in P_1\} \text{ and}$$

$$P_1 \gamma P_2 = \{q_1 \gamma q_2 \mid q_1 \in P_1, q_2 \in P_2\}.$$

We now discuss what constitutes a reasonable extension of a relational operator relative to this possibility function. However, first, we want the generalized operator to agree with the regular operator on Rel without regard to the possibility function.

Definition 7.8: Let γ be an operator on Rel and let γ' be an operator on $Rel\uparrow \cup Rel$. We say that γ' is *faithful* to γ if one of the following two conditions holds:

1. when γ and γ' are unary operators, $\gamma(r) = \gamma'(r)$ for every $r \in Rel$ for which $\gamma(r)$ is defined.

2. when γ and γ' are binary operators, $r \gamma q = r \gamma' q$ for every $r, q \in Rel$ for which $r \gamma q$ is defined.

Second, we would ideally like our generalized operator to give us the same set of possibilities as the standard operator.

Definition 7.9: Let γ be an operator on Rel and let γ' be an operator on $Rel\uparrow$. We say that γ' is a *precise* generalization of γ relative to possibility function $POSS$ if one of the following two conditions holds:

1. when γ and γ' are unary operators, $POSS(\gamma'(r)) = \gamma(POSS(r))$ for every $r \in Rel\uparrow$.
2. when γ and γ' are binary operators, $POSS(r \gamma' q) = POSS(r) \gamma POSS(q)$ for every $r, q \in Rel\uparrow$.

Unfortunately, not all relational operators have a precise generalization relative to $POSS$. Consider a join operator for $POSS$. It cannot be precise. For relations $r \in Rel\uparrow(R)$ and $q \in Rel\uparrow(Q)$, $POSS(r) \bowtie POSS(q)$ is subset of $SAT(\bowtie [R, Q])$. But, for some relation $p \in Rel\uparrow(RQ)$, $POSS(p)$ is not a subset of $SAT(\bowtie [R, Q])$. In these cases, we settle for a generalization of γ that captures everything in $\gamma(POSS(r))$ or $POSS(r) \gamma POSS(q)$ and as little extra as possible.

Definition 7.10: Let γ be an operator on Rel and let γ' be an operator on $Rel\uparrow$. We say that operator γ' is *adequate* for γ relative to possibility function

POSS if one of the following two conditions holds:

1. when γ and γ' are unary operators, $POSS(\gamma'(r)) \supseteq \gamma(POSS(r))$ for every $r \in Rel\uparrow$.
2. when γ and γ' are binary operators, $POSS(r \ \gamma' \ q) \supseteq POSS(r) \ \gamma \ POSS(q)$ for every $r, q \in Rel\uparrow$.

Furthermore, we say that operator γ' is *restricted* for γ relative to *POSS* if one of the following two conditions holds:

1. when γ and γ' are unary operators, for every $r \in Rel\uparrow$, there is no p in $Rel\uparrow$ such that $POSS(\gamma'(r)) \not\supseteq POSS(p) \supseteq \gamma(POSS(r))$.
2. when γ and γ' are binary operators, for every $r, q \in Rel\uparrow$, there is no p in $Rel\uparrow$ such that $POSS(r \ \gamma' \ q) \not\supseteq POSS(p) \supseteq POSS(r) \ \gamma \ POSS(q)$.

Clearly, if γ' is precise for γ , then γ' is adequate and restricted for γ .

We would also like the generalized operators to have properties that the standard operator possesses, such as commutativity or associativity. For example, if γ is an associative binary operator, we want a generalization γ' to satisfy

$$(p \ \gamma' \ q) \ \gamma' \ r = p \ \gamma' \ (q \ \gamma' \ r)$$

for $p, q, r \in Rel\uparrow$. Finally, we would like the generalized operators to return only minimal relations given minimal relations as input.

r_1		r_2		r_3			r_4	
employee	skill	employee	skill	employee	child	skill	employee	child
Smith	typing	Smith	ni	Smith	Sam	typing	Smith	Sam
Jones	filing	Jones	typing	Smith	Sue	typing	Smith	Sue
Jones	typing	ni	clerk	Jones	unk	ni	Jones	ni
Adams	ni	unk	dictation				unk	Joe
ni	dictation							

Figure 7-5. Some sample relations.

We now present generalizations for the standard operators, called null-union, null-difference, null-product, null-select, and null-project (denoted \cup' , $-'$, \times' , σ' , and π' , respectively), which are faithful, and at least adequate and restricted, if not precise. Some sample relations are shown in Figure 7-5. These will be used to illustrate the new operators.

7.1.2.1 Null-union

The null-union of two relations r on scheme R and q on scheme Q in $Rel \cup Rel \uparrow$ is a relation p on scheme P where:

1. $E_P = E_R \cup E_Q$, and
2. $p = r \cup' q = \{\hat{t} \mid t \tilde{\in} r \text{ or } t \tilde{\in} q\} = \{\hat{t} \mid t \in r \text{ or } t \in q\}$.

Some examples of null-union are shown in Figure 7-6.

Proposition 7.1: The operator null-union is faithful to standard union.

Proof: The only difference between the definition of null-union and standard union is that tuple set reduction is applied to the result of a null-union oper-

employee	skill
Smith	typing
Jones	filing
Jones	typing
Adams	ni
unk	dictation
ni	clerk

employee	child	skill
Smith	Sam	typing
Smith	Sue	typing
Jones	ni	filing
Jones	ni	typing
Jones	unk	ni
Adams	ni	ni
ni	ni	dictation

Figure 7-6. Examples of null-union.

ation. However, since we are dealing with relations in which there are no null values, this extra operation makes no changes. Thus, null-union is faithful to standard union. \square

Proposition 7.2: The operator null-union is a precise generalization of standard union with respect to possibility function *POSS*.

Proof: We show inclusion both ways. Let $p = r \cup' q$.

\supseteq Let $\hat{p} \in POSS(r) \cup POSS(q)$. There must be $\hat{r} \in POSS(r)$ and $\hat{q} \in POSS(q)$ such that $\hat{p} = \hat{r} \cup \hat{q}$. Let t_p be a tuple in p . Either $t_p \in r$ or $t_p \in q$. If $t_p \in r$, there is a tuple $t_{\hat{r}} \in \hat{r}$, and hence in \hat{p} , such that $t_{\hat{r}} \geq t_p$. A similar argument holds if $t_p \in q$. We conclude $\hat{p} \geq p$ and so $\hat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq POSS(r) \cup POSS(q)$.

\subseteq Let $\hat{p} \in POSS(p)$. Since $p \geq r$, $\hat{p} \geq r$ and so $\hat{p} \in POSS(r)$. Similarly, $\hat{p} \in POSS(q)$. Therefore, $\hat{p} \in POSS(r) \cup POSS(q)$, and so $POSS(p) \subseteq POSS(r) \cup POSS(q)$.

We conclude that null-union is a precise generalization of standard union for *POSS*. □

7.1.2.2 Null-difference

The null-difference of two relations r on scheme R and q on scheme Q in $Rel \cup Rel^\uparrow$ is a relation p on scheme P where $E_P = E_R \cup E_Q$ and

$$p = r -' q = \{t \mid t \tilde{\in} r \text{ and } t \not\tilde{\in} q\} = \{t \mid t \in r \text{ and } \forall s \in q : \neg(s \geq t)\}.$$

The definitions of null-union and null-difference were first proposed by Zaniolo [Zan2], who showed the given equivalences. The second equality is preferable as $\tilde{\in}$ implies a combinatorial explosion in generated tuples which are subsequently removed by *tuple set reduction*, and that *tuple set reduction* is not needed for difference as we assume the input relations are minimal.

Some examples of null-difference are shown in Figure 7-7. Null-difference is a faithful, and adequate and restricted generalization of standard difference. To show that null-difference is not precise, consider a relation r with one non-null tuple and an empty relation q . Every relation in $POSS(r -' q)$ must subsume r , whereas $POSS(r) - POSS(q)$ is empty. Thus, $POSS(r -' q) \neq POSS(r) - POSS(q)$.

Proposition 7.3: The operator null-difference is faithful to standard difference.

employee	skill
Smith	typing
Jones	filing
Adams	ni

employee	skill
unk	dictation
ni	clerk

Figure 7-7. Examples of null-difference.

Proof: When there are no null values then the only way for one tuple to subsume another is for them to be identical. Thus, in the definition of null-difference the statement $\forall s \in q : \neg(s \geq t)$ reduces to $\forall s \in q : \neg(s = t)$ which is equivalent to $\neg(t \in q)$. With this reduction, we have the standard definition of difference. Thus, null-difference is faithful to standard difference. \square

Proposition 7.4: The operator null-difference is an adequate and restricted generalization of standard difference with respect to possibility function *POSS*.

Proof: We show adequate and then restricted.

adequate: $POSS(r -' q) \supseteq POSS(r) - POSS(q)$.

Let $p = r -' q$, and $\hat{p} \in POSS(r) - POSS(q)$. Then, $\hat{p} \in POSS(r)$. Let t_p be a tuple in p . Then, t_p must be in r . Therefore, there is a tuple $t_{\hat{p}} \in \hat{p}$, such that $t_{\hat{p}} \geq t_p$. We conclude that $\hat{p} \geq p$ and so $\hat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq POSS(r) - POSS(q)$.

restricted: there does not exist p such that $POSS(r -' q) \supsetneq POSS(p) \supseteq POSS(r) - POSS(q)$.

Suppose there is some p . If $POSS(r -' q) \supsetneq POSS(p)$, then there must be some tuple t in p that does not subsume any tuple in $r -' q$. This means that the non-null valued attributes X of t do not match any tuple on X in $r -' q$. There are two possible reasons for this: either $t[X] \in r[X]$ and $\exists s \in q : s \geq t$, or $t[X] \notin r[X]$. In each case, any relation in $POSS(p)$ must contain a tuple which subsumes t , however, $POSS(r) - POSS(q)$ contains a relation which does not. In the first case, t 's possibility can be eliminated by the possibility of s in q that subsumes it, and in the second case, simply consider the possibilities of r that do not include a tuple which subsumes t . Therefore, $POSS(p) \not\subseteq POSS(r) - POSS(q)$, which is a contradiction.

We conclude that null-difference is an adequate and restricted generalization of standard difference for $POSS$. \square

We note that a generalized null-intersection operator is not derivable from null-difference alone. Figure 7-8 shows that the usual equivalence

$$r_1 \cap' r_2 = r_1 -' (r_1 -' r_2) = r_2 -' (r_2 -' r_1)$$

does not hold. However, as pointed out in [Zan2], the following more symmetric definition of intersection in terms of union and difference does carry forward to the null generalizations.

$$r_1 \cap' r_2 = (r_1 \cup' r_2) -' ((r_1 -' r_2) \cup' (r_2 -' r_1))$$

This result is also shown in Figure 7-8. Note that $r_1 -' r_2$, $r_2 -' r_1$, and $r_1 \cap' r_2$ now appropriately partition $r_1 \cup' r_2$ just as the standard operators do. We

$(r_1 \cup' r_2) -'$	
$((r_1 -' r_2) \cup' (r_2 -' r_1))$	
employee	skill
Jones	typing

$r_1 -' (r_1 -' r_2)$	
employee	skill
ni	dictation
Jones	typing

$r_2 -' (r_2 -' r_1)$	
employee	skill
Smith	ni
Jones	typing

Figure 7-8. Examples of null-intersection.

note also that null-intersection is an adequate and restricted generalization of standard intersection for *POSS*.

7.1.2.3 Null-product

The null-product of two relations is identical to the standard (cartesian) product, as no values are checked in the process. Let $r \in Rel(R) \cup Rel\uparrow(R)$ and $q \in Rel(Q) \cup Rel\uparrow(Q)$, with $E_R \cap E_Q = \emptyset$. Then the null-product of r and q is defined as follows:

$$r \times' q = \{t \mid \exists t_r \in r \text{ and } \exists t_q \in q, t[E_R] = t_r, \text{ and } t[E_Q] = t_q\}$$

Null-product is obviously a faithful and precise generalization of standard product.

7.1.2.4 Null-select

Selection of tuples comes in two flavors, comparison of an attribute value against a non-null constant and comparison of one attribute value against another. Let $r \in Rel(R) \cup Rel\uparrow(R)$ and let $A \in E_R$. Null selection is defined as follows:

$$\sigma'_{A \theta a}(r) = \{t \mid t \in r \text{ and } t[A] \theta a\}$$

$$\sigma'_{A \theta B}(r) = \{t \mid t \in r \text{ and } t[A] \theta t[B]\}$$

where θ is $=$ or $<$. Recall that null values are not equal to each other or to any other domain value, and with this stipulation null-select is essentially identical to standard select.

Null-select is faithful to standard select, but it is not precise. For $\sigma'_{A=a}$, note that for any relation $q \in \sigma_{A=a}(POSS(r))$, every tuple $t \in q$ has $t[A] = a$. For any relation p , $POSS(p)$ contains relations whose tuples are not all a on A . However, the definitions are adequate and restricted.

Proposition 7.5: The operator null-select is faithful to standard select.

Proof: As the definitions of null-select and select are identical when no null values are present, the result follows immediately. \square

Proposition 7.6: The operator null-select is an adequate and restricted generalization of standard select with respect to possibility function $POSS$.

Proof: We show adequate and then restricted. Let F be any selection predicate.

adequate: $POSS(\sigma'_F(r)) \supseteq \sigma_F(POSS(r))$.

Let $p = \sigma'_F(r)$, and $\hat{p} \in \sigma_F(POSS(r))$. There must be $\hat{r} \in POSS(r)$ such that $\hat{p} = \sigma_F(\hat{r})$. Let t_p be a tuple in p . Then, t_p must be in r and satisfy F . Therefore, there is a tuple $t_{\hat{p}} \in \hat{r}$, such that $t_{\hat{p}} \geq t_p$, and satisfies F . We conclude that $\hat{p} \geq p$ and so $\hat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq \sigma_F(POSS(r))$.

restricted: there does not exist p such that $POSS(\sigma'_F(r)) \supsetneq POSS(p) \supseteq \sigma_F(POSS(r))$.

Suppose there is some p . If $POSS(\sigma'_F(r)) \supsetneq POSS(p)$, then there must be some tuple t in p that does not subsume any tuple in $\sigma'_F(r)$. This means that the non-null valued attributes X of t do not match any tuple on X in $\sigma'_F(r)$. There are two possible reasons for this: either $t[X] \in r[X]$ and t does not satisfy F , or $t[X] \notin r[X]$. In each case, any relation in $POSS(p)$ must contain a tuple which subsumes t , however, $\sigma_F(POSS(r))$ contains a relation which does not. In the first case, t 's possibility is eliminated by applying the selection predicate, and in the second case, simply consider the possibilities of r that do not include a tuple which subsumes t . Therefore, $POSS(p) \not\supseteq \sigma_F(POSS(r))$, which is a contradiction.

We conclude that null-select is an adequate and restricted generalization of standard select for $POSS$. □

7.1.2.5 Null-project

While standard projection eliminates duplicate tuples from the reduced relation, null-projection eliminates less informative tuples. Let $r \in Rel(R) \cup Rel\uparrow(R)$ and let $A_1, A_2, \dots, A_n \in E_R$. Null-project is defined as follows:

$$\pi'_{A_1, A_2, \dots, A_n}(r) = \{t[A_1 A_2 \dots A_n] \mid t \in r\}$$

Examples of null-project are shown in Figure 7-9.

skill
dictation
filing
typing

child	skill
Sam	typing
Sue	typing
ni	filing
ni	dictation

Figure 7-9. Examples of null-project.

Proposition 7.7: The operator null-project is faithful to standard project.

Proof: As in the proof of Proposition 7.1, without null values, tuple set reduction has no affect on the result of the relation, making the definitions of null-project and standard project identical. \square

Proposition 7.8: The operator null-project is a precise generalization of standard project with respect to possibility function *POSS*.

Proof: We show inclusion both ways. Let $p = \pi'_X(r)$, where X are the attributes being projected.

\supseteq Let $\hat{p} \in \pi_X(POSS(r))$. There must be $\hat{r} \in POSS(r)$ such that $\hat{p} = \pi_X(\hat{r})$. Let t_p be a tuple in p . Then, $t_p \in r[X]$. Since $t_p \in r[X]$, there is a tuple $t_{\hat{p}} \in \hat{r}[X]$, and hence in \hat{p} , such that $t_{\hat{p}} \geq t_p$. We conclude $\hat{p} \geq p$ and so $\hat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq \pi_X(POSS(r))$.

\subseteq Let $\hat{p} \in POSS(p)$. Consider each tuple $t_p \in p$. Each t_p is in $r[X]$ and possibly eliminated some other tuples in $r[X]$ since t_p subsumed them. We then construct the following relation in $POSS(r)$: for each

tuple in r whose projection is t_p and the tuples in r whose projection it subsumes, make the same assignment to the null values in attributes X that was made in constructing \hat{p} . By making the same assignment, in the projection $\pi_X(POSS(r))$, the tuples which were subsumed in the null-project will be duplicates and thus, eliminated in both cases. Therefore, $\hat{p} \in \pi_X(POSS(r))$, and so $POSS(p) \subseteq \pi_X(POSS(r))$.

We conclude that null-project is a precise generalization of standard project for *POSS*. □

7.1.2.6 Join

As in the case of the standard operators, the various θ -joins can be defined as selections on a cartesian product. In our case,

$$r \bowtie'_{A\theta B} q = \sigma'_{A\theta B}(r \times' q).$$

As in [Zan2, LaP], the use of null values allows the definition of new information-preserving joins (also called *outer joins*) which include tuples that normally do not participate in the join. An information-preserving equijoin is defined by

$$(r \bowtie'_{A=B} q) \cup' r \cup' q.$$

Figure 7-10 shows an example of the information-preserving equijoin of r_2 and r_4 .

$$(r_2 \underset{\text{employee=employee}}{\bowtie} r_4) \cup r_2 \cup r_4$$

r_2 .employee	skill	r_4 .employee	child
Smith	ni	Smith	Sam
Smith	ni	Smith	Sue
Jones	typing	Jones	ni
ni	clerk	ni	ni
unk	dictation	ni	ni
ni	ni	unk	Joe

Figure 7-10. Information preserving equijoin.

7.2 Introducing Null Values into the \neg 1NF Relational Model

Previous research on nulls in \neg 1NF relations has been either ambiguous or incompletely treated. One source of concern is the effect of the *unnest* operator on empty sets. As defined in Chapter 4 *unnest* produces a flatter relation structure with each element of the unnested set forming a value in a separate tuple in the flatter relation. When the set is empty, it is not clear what this operation means. Schek states, "In the general case *unnest* on empty relations will produce undefined attribute values" [Sch2;180]. However, if the empty set has a meaning in the relation, then whatever it unnests to should have meaning also. In the VERSO model [AB1], empty sets are used as null values for set-valued attributes. However, nulls are not allowed for atomic-valued attributes. Thus, when an empty set is unnested the entire tuple is deleted from the resulting relation.

Two researchers have assigned the *non-existent* interpretation to empty set. One of Makinouchi's properties of "not-necessarily-normalized" relations is that "A null set (\emptyset) may be in the domain of a relation column. \emptyset means exactly non-existence" [Mak;448]. In deriving an extended set-containment operation for 1NF relations with non-existent nulls, Zaniolo [Zan1] discusses the \neg 1NF viewpoint. In this development, he assigns the non-existence meaning to the empty set, viewing the non-existent null as the image of an empty set when mapping from an unnormalized relation to a normalized one.

We believe that the correct interpretation for empty set is the *no-information* one. We have already seen in the definition of *tuple set reduction* that the null tuple is eliminated from any relation even if it is the only tuple in the relation. So, in the simplest case of a relation with one attribute, we have that the empty relation is equivalent to the relation containing only the tuple $\langle ni \rangle$. This is consistent with the open world assumption we have been making in which we do not assume that the empty relation indicates that no tuples belong in the relation but that we currently have no information about the world and so we do not know if the tuples belong or not. As we will see, this means an empty nested relation should unnest to a no-information, null tuple.

7.2.1 Basic Concepts

When nulls are introduced into our model, the concept of *more informative* (or *subsumes*) must be extended to handle nested relations. The main idea is to treat nested relations as values which must be more informative than the corresponding nested relation in the less informative tuple. In addition, a *null tuple* which consists of all ni values in the 1NF model is extended in the \neg 1NF model so that all zero order attributes have ni values and all higher order attributes are empty or, equivalently, contain exactly one null tuple. Thus, our new definition of *more informative*, which includes the old one as a special case, is as follows.

Definition 7.11: Let t_1 be a tuple on zero order attributes X_1 and higher order attributes Y_1 , and let t_2 be a tuple on zero order attributes X_2 and higher order attributes Y_2 . The tuple t_1 is said to be *more informative* than the tuple t_2 when:

- (a) for each $B \in X_2$, if $t_2[B]$ is not ni then $B \in X_1$,
- (b) for each $C \in Y_2$, if $t_2[C]$ contains a tuple that is not null then $C \in Y_1$,
- (c) for each $A \in X_1 \cap X_2$, $glb(t_1[A], t_2[A]) = t_2[A]$, and
- (d) for each $D \in Y_1 \cap Y_2$ and tuple $u_2 \in t_2[D]$, there exists some tuple $u_1 \in t_1[D]$ which is *more informative* than u_2 .

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

Figure 7-11. A sample relation on the Emp scheme.

Example 7.3: Recall the Emp scheme and sample relation (shown again in Figure 7-11) introduced in Chapter 5. If a new employee, say Jones, is added to the database and we do not know anything about him except his name, then we would add the tuple $\langle \text{Jones}, \{\}, \{\} \rangle$, or, equivalently, $\langle \text{Jones}, \{\langle \text{ni}, \text{ni} \rangle\}, \{\langle \text{ni}, \{\langle \text{ni}, \text{ni} \rangle\}\} \rangle$. If we find out later that Jones has no children and has some skill for which he took a 1981 exam, we could update the tuple to $\langle \text{Jones}, \{\text{dne}, \text{dne}\}, \{\langle \text{unk}, \{\langle 1981, \text{unk} \rangle\}\} \rangle$. \square

There is an aspect of our definition of *more informative* which goes beyond nulls. Consider the following tuple

$$\langle \text{Smith}, \{\langle \text{Sam}, 2/10/84 \rangle\}, \{\langle \text{ni}, \{\langle \text{ni}, \text{ni} \rangle\}\} \rangle.$$

According to Definition 7.11, this tuple is less informative than the one in Figure 7-11. Note that the Children attribute in the original "Smith" tuple is a nested relation with two tuples while in the new tuple only one of the Children tuples

exists. This reasoning stems from our interpretation of the relationship between the attributes in $\neg 1NF$ relations. Nested relations are *not* nondecomposable values, so that it is the tuples of the nested relation that are related to the other attributes. Thus an employee is related to each child and there is no particular significance to sets of children. Similar reasoning about the significance of sets led to our definition of PNF. However, the requirement of PNF is a somewhat different notion than that of subsumption, as the following example shows.

Example 7.4: Let $t_1 = \langle \text{Smith}, \{ \langle \text{Sam} \rangle, \langle \text{Sue} \rangle \} \rangle$ and $t_2 = \langle \text{Smith}, \{ \langle \text{Sue} \rangle, \langle \text{Bill} \rangle \} \rangle$ be tuples from a projected employee relation. We have that $t_1 \not\preceq t_2$ and $t_2 \not\preceq t_1$, but under PNF t_1 and t_2 would be combined into $t_3 = \langle \text{Smith}, \{ \langle \text{Sam} \rangle, \langle \text{Sue} \rangle, \langle \text{Bill} \rangle \} \rangle$. \square

The definitions of *x-element* ($\tilde{\epsilon}$), and *tuple set reduction* ($\hat{\{set\ of\ tuples\}}$), from section 7.1, carry over to $\neg 1NF$ in a straightforward manner. However, the *meet* of two $\neg 1NF$ tuples must be extended to handle nested relations. This can be done using the *glb* function for zero order attributes and applying the definition recursively for higher order attributes.

Definition 7.12: Let U be the attributes on which two tuples t_1 and t_2 are defined, where t_1 and t_2 have been extended to U with the addition of nil values for zero order attributes and single null tuple relations for higher order attributes, if necessary. A tuple t is the *meet* of t_1 and t_2 , written $t_1 \wedge t_2$, when for each zero order attribute $A \in U$, $t[A] = glb(t_1[A], t_2[A])$, and for each higher

order attribute $X \in U$, $t[X] = \{s \wedge u \mid s \in t_1[X] \text{ and } u \in t_2[X]\}$.

Finally, the ideas of more informative relations, information-wise equivalence and minimal representations for a relation all have the same definitions when we substitute the $\neg 1NF$ version of subsumption.

7.2.2 Operators for $\neg 1NF$ Relations with Nulls

Since the mapping between $1NF$ and $\neg 1NF$ relations is an important one, we need to revise the definitions of *nest* and *unnest* to deal with the presence of null values. For *nest*, we deal with the problem of null values for the partitioning attributes (the attributes not being nested), and for *unnest* we deal with subsumption and possible loss of information. Once this is dealt with, we provide, where possible, precise extensions to the $\neg 1NF$ operators defined in Chapter 5 accommodating null values. Once again we will work only on relations in PNF. However, our definition of PNF relies on the definition of functional dependency in which we test equality of attribute values, and therefore, we need to specify how null values should be treated. For purposes of testing for equality, $ni \neq ni$, $unk \neq unk$, and $dne = dne$. The intuition behind this will be discussed in what follows.

7.2.2.1 Null-nest

When null values occur as values of attributes which are being nested, then no special rules need apply. We could use *tuple set reduction* on each nested

relation, but if we assume that the input relation is minimal then the new relation and its new nested relations will all be minimal as well. Problems in the standard definition of *nest* arise when nulls are values of the partitioning attributes. The question is whether we equate nulls for partitioning purposes. At first glance, equating nulls would be advantageous in that we could have a succinct notation for grouping all values for which we do not have a fully defined partition value. However, doing this grouping would give the impression that one value could replace the null for all members of the group. Since this is not generally true, we should not equate *no-information* and *unknown* nulls, when partitioning the relation. The *does not exist* null is a special case though. Since there is no value which can replace a dne null, it is appropriate to nest all tuples which have that property together. Thus, our definition of *null-nest* is not different from standard nest except that two attribute values are considered equal iff they are both the same domain value or they are both dne nulls.

Example 7.5: Consider the 1NF relation of Figure 7-12a. Suppose that we want to nest all courses taught by each teacher. For the two "Smith" tuples the standard nest applies and we get the single tuple with "Math1" and "Math2" together in a nested relation. The same applies to the two tuples with dne nulls. These two tuples indicate that "Math5" and "Math6" are courses that exist, but there are no teachers teaching them, so we can group these courses together as courses for which there is no teacher. If we find that our information was wrong and "Math5" does have a teacher then we would be forced to update this tuple

r	
teacher	course
Smith	Math1
Smith	Math2
dne	Math5
dne	Math6
ni	Science1
ni	Science2

(a)

$\nu_{course=(course)}^r$	
teacher	course*
	course
Smith	Math1
	Math2
dne	Math5
	Math6
ni	Science1
ni	Science2

(b)

Figure 7-12. Example of *nest* with null values.

just as if we found out the “Smith” is not really teaching “Math2”. Finally, the two tuples with *ni* nulls are nested singly, since we have no assurance that they will be in the same partition when more information is found out about them. In this case, the two courses may be newly added ones, for which we know nothing about who will teach them or even if they will be taught. Figure 7-12b shows the nested relation. □

Before we consider the preciseness of the null-nest operator, we introduce a modified possibility function to deal with PNF relations. Consider the nested relation of Example 7.5. Using our current definition of *POSS*, one possibility for this relation is constructed by replacing the *ni* nulls with the same value, say “Jones.” As a result, we no longer have a PNF relation. An alternative possibility, representing the same information, is constructed by replacing the $\langle ni, \{\langle Science1 \rangle\} \rangle$ and $\langle ni, \{\langle Science2 \rangle\} \rangle$ tuples with the single

tuple, $\langle \text{Jones}, \{ \langle \text{Science1} \rangle, \langle \text{Science2} \rangle \} \rangle$. This possibility also satisfies the current definition of *POSS*, but the resulting relation is in PNF. Therefore, we will use a modified definition of *POSS*, so that only PNF relations are allowed. The set of *PNF possibilities* for relation r on scheme R is denoted $POSS^*(r)$, and is defined as:

$$POSS^*(r) = \{q \mid q \in Rel^*(R) \cup Rel(R) \text{ and } q \geq r \text{ and } q \text{ is in PNF}\}.$$

Proposition 7.9: Null-nest is a precise generalization of standard nest with respect to PNF possibility function $POSS^*$.

Proof: Let X be the attributes of r being nested. We show that $POSS^*(\nu'_{B=(X)}(r)) = \nu_{B=(X)}(POSS^*(r))$. We show inclusion both ways. Let $p = \nu'_{B=(X)}(r)$.

\subseteq Let $\hat{p} \in POSS^*(p)$. There are two cases depending on the assignment by $POSS^*$ to null values in the partition keys of p . In the first case, if $POSS^*$ assigned the same value to nulls in otherwise equal partition keys of p , then these tuples will be combined by the PNF requirement of $POSS^*$. By making this same assignment of nulls directly to r , then nesting will also combine these tuples. In the second case, if we make the same assignment to nulls in \hat{p} and in r , then nesting on $POSS^*(r)$ will also produce \hat{p} . Thus, $\hat{p} \in \nu_{B=(X)}(POSS^*(r))$.

\supseteq Let $\hat{p} \in \nu_{B=(X)}(POSS^*(r))$. There must be $\hat{r} \in POSS^*(r)$ such that

$\hat{p} = \nu_{B=(X)}(\hat{r})$. Consider the assignment of values made by $POSS^*$ in \hat{r} . If we, in $POSS^*(p)$, make the same assignment to the corresponding nulls in p , then we get also \hat{p} . Thus, $\hat{p} \in POSS^*(p)$.

We conclude that null-nest is a precise generalization of standard nest for $POSS^*$. □

7.2.2.2 Null-unnest

If nested relations are inserted into our database solely by application of the nest operator to relations in 1NF, then the standard definition of unnest can apply to relations with nulls and there are no problems. However, if we allow arbitrary nested relations then unnesting can produce non-minimal relations and cause loss of information.

Example 7.6: Recalling the database scheme of the previous example, consider a relation r with two tuples $t_1 = \langle \text{Jones}, \{ \langle \text{Math} \rangle, \langle \text{Science} \rangle \} \rangle$ and $t_2 = \langle \text{ni}, \{ \langle \text{Math} \rangle, \langle \text{English} \rangle \} \rangle$. If we unnest r , then the resulting $\langle \text{ni}, \text{Math} \rangle$ tuple is less informative than the $\langle \text{Jones}, \text{Math} \rangle$ tuple. Thus, even though t_1 and t_2 form a minimal relation, their unnested counterparts do not. □

The problem with arbitrary \neg 1NF relations is they allow the misuse of ni and unk nulls in the partition attributes. Our previous discussion of the nest operator showed that when an ni or a unk null is in one of the partition attributes, then the nested relation should have cardinality of one. But, one

can argue that we may know that, say, two tuples are both related to one undetermined value and we should take advantage of that fact and store those two tuples in the same nested relation. If this is true, then an answer is to use *marked ni* and *unk* nulls [Sci2]. Then a tuple can be subsumed only if its marked nulls do not exist in any tuple other than the subsuming tuple. Using marked nulls also avoids some loss of information. In the previous example, if we unnest r and then perform the reverse nest operation, we would find three tuples in the result as the tuples with *ni* as the teacher value would not be nested together as per our previous arguments. It would be appropriate to equate identical marked nulls and so a nest would return the original relation.

Another reason for our treatment of *ni* and *unk* is so that null-unnest is a precise generalizations of the standard operator. In Example 7.6, every relation in $\mu_{courses}(POSS^*(r))$ must contain $\langle x, \text{Math} \rangle$ and $\langle x, \text{English} \rangle$ for some value x . However, there are relations in $POSS^*(\mu'_{courses}(r))$ which do not have both of these tuples for some value x . So, under the assumption that tuples with *ni* or *unk* nulls in the partition attributes of a relation (nested or otherwise) have only single tuple nested relations for each higher order attribute, our definition of null-unnest is unchanged from the standard unnest definition. Furthermore, we can prove that null-unnest is a precise generalization.

Proposition 7.10: Null-unnest is a precise generalization of standard unnest with respect to PNF possibility function $POSS^*$.

Proof: We show that $POSS^*(\mu'_B(r)) = \mu_B(POSS^*(r))$. Let $p = \mu'_B(r)$.

\subseteq Let $\hat{p} \in POSS^*(p)$. If we make the same assignment to the nulls in p as in the nested relation r then $\hat{p} \in \mu_B(POSS^*(r))$. This is possible since we assume that tuples in r with null values in the partition keys have single tuple nested relations. Therefore, there is a one-to-one correspondence between these null values in both r and p .

\supseteq Let $\hat{p} \in \mu_B(POSS^*(r))$. Then there must be $\hat{r} \in POSS^*(r)$ such that $\hat{p} = \mu_B(\hat{r})$. Let t_p be a tuple in p . Now, t_p unnested from some tuple t_r in r , which has some PNF possibility $t_{\hat{r}} \in \hat{r}$ such that $t_{\hat{r}} \geq t_r$. Let $t_{\hat{p}} = \mu_B(t_{\hat{r}})$. Then, we have $t_{\hat{p}} \geq t_p$. We conclude that $\hat{p} \geq p$ and so $\hat{p} \in POSS^*(p)$.

We conclude that null-unnest is a precise generalization of standard unnest for $POSS^*$. □

With this result we can now show that the null-unnest* operator (μ'^*) is a precise generalization of the standard unnest* operator.

Corrolary 7.1: Null-unnest* is a precise generalization of standard unnest* with respect to PNF possibility function $POSS^*$.

Proof: Apply the same argument as for Proposition 7.10, only use complete unnesting instead of single unnesting. □

7.2.3 Null-extended Operators

Let $Rel\uparrow^*$ represent the set of all relations which are not in 1NF or which contain a null value. Thus, $Rel^* \cup Rel\uparrow = Rel\uparrow^*$ and $Rel \cap Rel\uparrow^* = \emptyset$. Our goal is to generalize the \neg 1NF operators to deal with null values. We have two choices for our definition of a precise generalization for the operators. We can either apply the PNF possibility function first and then unnest the result or we can unnest first and then apply the PNF possibility function, resulting in the following two definitions.

Definition 7.13: Let γ be an operator on Rel and let γ'^* be an operator on $Rel\uparrow^*$. We say that γ'^* is a *precise* generalization of γ relative to unnesting and PNF possibility function $POSS^*$ if one of the following two conditions holds:

1. when γ and γ'^* are unary operators, $\mu^*(POSS^*(\gamma'^*(r))) = \gamma(\mu^*(POSS^*(r)))$ for every $r \in Rel\uparrow^*$ for which $\gamma'^*(r)$ is defined.
2. when γ and γ'^* are binary operators, $\mu^*(POSS^*(r \ \gamma'^* \ q)) = \mu^*(POSS^*(r)) \ \gamma \ \mu^*(POSS^*(q))$ for every $r, q \in Rel\uparrow^*$ for which $r \ \gamma'^* \ q$ is defined.

Definition 7.14: Let γ be an operator on Rel and let γ'^* be an operator on $Rel\uparrow^*$. We say that γ'^* is a *precise* generalization of γ relative to unnesting and PNF possibility function $POSS^*$ if one of the following two conditions holds:

1. when γ and γ'^* are unary operators, $POSS^*(\mu'^*(\gamma'^*(r))) =$

$\gamma(POSS^*(\mu'^*(r)))$ for every $r \in Rel\uparrow^*$ for which $\gamma'^*(r)$ is defined.

2. when γ and γ'^* are binary operators, $POSS^*(\mu'^*(r \ \gamma'^* \ q)) = POSS^*(\mu'^*(r)) \ \gamma \ POSS^*(\mu'^*(q))$ for every $r, q \in Rel\uparrow^*$ for which $r \ \gamma'^* \ q$ is defined.

Theorem 7.1: Definition 7.13 and Definition 7.14 are equivalent.

Proof: By Corrolary 7.1, we know that null-unnest* is a precise generalization of standard unnest* for $POSS^*$. Thus, the definitions are equivalent. \square

There are corresponding definitions of *adequate* and *restricted* for $Rel\uparrow^*$, and there are three specifications of faithfulness we could use: comparing relations in $Rel\uparrow^*$ to relations in Rel , $Rel\uparrow$, and Rel^* . As in the previous section, proofs of faithfulness are straightforward and so we shall omit them here.

7.2.3.1 Null-extended union

Our definition of null-extended union can be revised to accommodate nulls by adding tuple set reduction as follows.

Definition 7.15: In order to take the *null-extended union* of two relations r_1 and r_2 we require that they have equal relation schemes, say R . The scheme of the resultant structure is also R . We define *null-extended union* at the instance level as follows. Let X range over the zero order attributes in E_R and Y range

over the higher order attributes in E_R . The *null-extended union* of r_1 and r_2 is:

$$\begin{aligned}
 r_1 \cup^{e'} r_2 = \widehat{t} \mid & (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 : (\forall X, Y \in E_R : t[X] = t_1[X] = t_2[X] \\
 & \wedge t[Y] = (t_1[Y] \cup^{e'} t_2[Y]))) \\
 \vee & (t \in r_1 \wedge (\forall t' \in r_2 : (\forall X \in E_R : t[X] \neq t'[X]))) \\
 \vee & (t \in r_2 \wedge (\forall t' \in r_1 : (\forall X \in E_R : t[X] \neq t'[X]))) \widehat{t}
 \end{aligned}$$

Note, this definition is recursive in that we apply the *null-extended union* to each higher order attribute Y .

As for extended union (section 5.2.1), we require the use of the Δ -union operator which maintains the join dependency involving the path set of $\neg 1NF$ relation's scheme tree.

Proposition 7.11: Null-extended union is a precise generalization of Δ -union with respect to unnesting and PNF possibility function $POSS^*$, where the join dependency used in the Δ -union is the path set of the $\neg 1NF$ relation's scheme tree.

Proof: We show that $\mu^*(POSS^*(r \cup^{e'} q)) = \mu^*(POSS^*(r)) \cup^\Delta \mu^*(POSS^*(q))$. By Proposition 5.3, we know that extended union is a precise generalization of Δ -union, and so $\mu^*(POSS^*(r)) \cup^\Delta \mu^*(POSS^*(q)) = \mu^*(POSS^*(r) \cup^e POSS^*(q))$. Thus, we only need to show that $POSS^*(r \cup^{e'} q) = POSS^*(r) \cup^e POSS^*(q)$. We show inclusion both ways. Let $p = r \cup^{e'} q$.

\supseteq Let $\hat{p} \in POSS^*(r) \cup^e POSS^*(q)$. There must be $\hat{r} \in POSS^*(r)$ and $\hat{q} \in POSS^*(q)$ such that $\hat{p} = \hat{r} \cup^e \hat{q}$. Let t_p be a tuple in p . Either

$t_p \in r$, $t_p \in q$, or t_p is a combination of tuples in r and q with equal partition keys. If $t_p \in r$, there is a tuple $t_{\hat{p}} \in \hat{r}$ such that $t_{\hat{p}} \geq t_p$. Now, $t_{\hat{p}}$ is either in \hat{p} or is included in a combined tuple of \hat{p} , since the null values of some partition key may have been assigned values that make the partition key non-unique. In any case, this tuple subsumes t_p . A similar argument can be made if $t_p \in q$. If t_p is a combination of tuples in r and q , then there are no null values in the outer most partition key. Therefore, in \hat{p} , these tuples will also combine, and there is a possibility which subsumes t_p . We conclude $\hat{p} \geq p$, and so $\hat{p} \in POSS^*(p)$. Therefore, $POSS^*(p) \supseteq POSS^*(r) \cup^c POSS^*(q)$.

\subseteq Let $\hat{p} \in POSS^*(p)$. Since $p \geq r$, $\hat{p} \geq r$ and \hat{p} is in PNF. Therefore, $\hat{p} \in POSS^*(r)$. Similarly, $\hat{p} \in POSS^*(q)$. Then, $\hat{p} \in POSS^*(r) \cup^c POSS^*(q)$, and so $POSS^*(p) \subseteq POSS^*(r) \cup^c POSS^*(q)$.

We conclude that null-extended union is a precise generalization of Δ -union for $POSS^*$ with respect to unnesting. \square

7.2.3.2 Null-extended difference

We change the definition of extended difference to include null values by keeping tuples in a relation only if they are not subsumed by some tuple in the other relation.

Definition 7.16: Let r_1 and r_2 be relations on scheme R . Let X range over the

zero order attributes in E_R and Y and Z range over the higher order attributes in E_R . The *null-extended difference* of r_1 and r_2 is:

$$r_1 -^{e'} r_2 = \{t \mid (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 \wedge \exists Z \in E_R : \\ (\forall X, Y \in E_R : t[X] = t_1[X] = t_2[X] \wedge t[Y] = (t_1[Y] -^{e'} t_2[Y]))) \\ \vee (t \in r_1 \wedge (\forall t' \in r_2 : \neg(t' \geq t)))\}$$

Proposition 7.12: Null-extended difference is an adequate and restricted generalization of Δ -difference with respect to unnesting and possibility function $POSS^*$, where the join dependency used in the Δ -difference is the path set of the $\neg 1NF$ relation's scheme tree.

Proof: We show adequate and then restricted.

adequate: $\mu^*(POSS^*(r -^{e'} q)) \supseteq \mu^*(POSS^*(r)) - \mu^*(POSS^*(q))$.

By Proposition 5.8, we know that extended difference is a precise generalization of Δ -difference, and so $\mu^*(POSS^*(r)) -^\Delta \mu^*(POSS^*(q)) = \mu^*(POSS^*(r) -^e POSS^*(q))$. Thus, we need only show that $POSS^*(r -^{e'} q) \supseteq POSS^*(r) -^e POSS^*(q)$. Let $p = r -^{e'} q$, and $\hat{p} \in POSS^*(r) -^e POSS^*(q)$. Then, there exists $\hat{r} \in POSS^*(r)$ and $\hat{q} \in POSS^*(q)$, such that $\hat{p} = \hat{r} -^e \hat{q}$. Let t_p be a tuple in p . Then, t_p must be in r with, perhaps, some of its nested relations reduced by interaction with a tuple t_q in q . Therefore, there must be tuples $t_{\hat{r}} \in \hat{r}$ and $t_{\hat{q}} \in \hat{q}$ which will also interact in the same way, noting that interaction occurs only when the zero order attributes have non-null values. Thus there is a tuple $t_{\hat{p}} = t_{\hat{r}} -^e t_{\hat{q}}$

in \hat{p} , such that $t_{\hat{p}} \geq t_p$. We conclude that $\hat{p} \geq p$ and so $\hat{p} \in POSS^*(p)$.

Therefore, $POSS^*(p) \supseteq POSS^*(r) -^e POSS^*(q)$.

restricted: there does not exist p such that $\mu^*(POSS^*(r -^{e'} q)) \not\supseteq \mu^*(POSS^*(p)) \supseteq \mu^*(POSS^*(r)) -^\Delta \mu^*(POSS^*(q))$.

As in the case for adequate, we need only show that there does not exist p such that $POSS^*(r -^{e'} q) \not\supseteq POSS^*(p) \supseteq POSS^*(r) -^e POSS^*(q)$. Suppose there is some p . If $POSS^*(r -^{e'} q) \not\supseteq POSS^*(p)$, then there must be some tuple t in p that does not subsume any tuple in $r -^{e'} q$. This means that the non-null valued zero order attributes X of t , or some nested relation in t , do not match any tuple on X in the corresponding place in $r -^{e'} q$. Let z be the relation (either r or a nested relation in r) and t' the tuple in z where the matching does not occur, and w be the corresponding, possibly empty, relation in q . There are two possible reasons for there not being a match: either $t'[X] \in z$ and $\exists s \in w : s \geq t'$, or $t'[X] \notin z[X]$. In each case, the corresponding relation in $POSS^*(p)$ must contain a tuple which subsumes t' , however, $POSS^*(r) - POSS^*(q)$ contains a relation in which the corresponding relation does not. In the first case, the possibility of t' can be eliminated by the possibility of s in w that subsumes it, and in the second case, simply choose not to include t' in $POSS^*(r)$. Therefore, $POSS^*(p) \not\supseteq POSS^*(r) - POSS^*(q)$, which is a contradiction.

We conclude that null-extended difference is an adequate and restricted gener-

alization of Δ -difference for $POSS^*$ with respect to unnesting. \square

7.2.3.3 Intersection, Cartesian Product, and Select

We will not formally define these "null-extended" versions of these operators.

A null-extended intersection can be obtained from union and difference by

$$r_1 \cap^{e'} r_2 = (r_1 \cup^{e'} r_2) -^{e'} ((r_1 -^{e'} r_2) \cup^{e'} (r_2 -^{e'} r_1)).$$

We note also that null-extended intersection is an adequate and restricted generalization of standard intersection with respect to unnesting and PNF possibility function $POSS^*$. For select we will use *null-select* as defined in section 7.1, and the standard cartesian product operator.

7.2.3.4 Join

The problems involved in defining join operations for relations with nulls and for $\neg 1NF$ relations have been discussed before. Combining nulls and $\neg 1NF$ does not improve the situation. However, our limited operator, extended natural join, does have an adequate and restricted generalization with respect to PNF possibility function $POSS^*$.

Definition 7.17: Let X be the higher order attributes in $E_{R_1} \cap E_{R_2}$, $A = E_{R_1} - X$, and $B = E_{R_2} - X$. Then the *null-extended natural join* is $r_1 \bowtie^{e'} r_2$ which produces a relation r on scheme R where:

1. $R = (A, X, B)$, and

$$2. r = \{t \mid (\exists u \in r_1, v \in r_2 : t[A] = u[A] \wedge t[B] = v[B] \wedge t[X] = (u[X] \cap^{e'} v[X]) \wedge t[X] \neq \emptyset)\}$$

Note we use *null-extended intersection* to combine the nested relations, and that zero order attributes can only have equal values if neither is ni or unk.

Proposition 7.13: Null-extended natural join is an adequate and restricted generalization of standard natural join with respect to unnesting and PNF possibility function $POSS^*$.

Proof: By Proposition 5.10, we know that extended natural join is a precise generalization for standard natural join. Therefore, we need only show that null-extended natural join is an adequate and restricted generalization of extended natural join. We show adequate and then restricted.

adequate: $POSS^*(r \bowtie^{e'} q) \supseteq POSS^*(r) \bowtie^e POSS^*(q)$.

Let $p = r \bowtie^{e'} q$ and $\hat{p} \in POSS^*(r) \bowtie^e POSS^*(q)$. Also, let C be the common zero order attributes of r and q . Then, there must be $\hat{r} \in POSS^*(r)$ and $\hat{q} \in POSS^*(q)$ such that $\hat{p} = \hat{r} \bowtie^e \hat{q}$. Let t_p be a tuple in p . Then, there are tuples $t_r \in r$ and $t_q \in q$ such that $t_p[C] = t_r[C] = t_q[C]$. There are also tuples $t_{\hat{r}} \in \hat{r}$ and $t_{\hat{q}} \in \hat{q}$ that agree on C with t_p , and will participate in the join giving $t_{\hat{p}}$. Now, the common higher order attributes X of $t_{\hat{r}}$ and $t_{\hat{q}}$ will participate in an extended intersection, the result of which will subsume the result of the null-extended intersection of $t_r[X]$ and $t_q[X]$.

Therefore, $t_{\hat{p}} \geq t_p$, $\hat{p} \geq p$, and so $\hat{p} \in POSS^*(p)$.

restricted: there does not exist p such that $POSS^*(r \bowtie^{e'} q) \supsetneq POSS^*(p) \supseteq POSS^*(r) \bowtie^e POSS^*(q)$.

Suppose there is some p . If $POSS^*(r \bowtie^{e'} q) \supsetneq POSS^*(p)$, then there must be some tuple t in p that does not subsume any tuple in $r \bowtie^{e'} q$. Thus, t contains non-null values which must occur in any possibility of p , but not in all possibilities of $r \bowtie^{e'} q$. Consider the possibilities for tuples in r and q which could exist to join to make a possibility for t . Since t does not subsume any tuple in $r \bowtie^{e'} q$, it must either have projections on the common zero order attributes that are null or different actual values, or have different actual values in a common nested relation. In the first case, there is a possibility for tuples in r and q which set the null value to different actual values, and so they do not participate in the join. In the second and third case, every possibility of p must contain those different values, yet there are possibilities of r and q which do not. Therefore, there is a possibility of r and q whose extended join is not a possibility of p . So, $POSS^*(p) \not\supseteq POSS^*(r) \bowtie^e POSS^*(q)$, which is a contradiction.

We conclude that null-extended natural join is an adequate and restricted generalization of standard natural join with respect to unnesting and PNF possibility function $POSS^*$.

7.2.3.5 Null-extended Projection

We define null-extended projection as an extended projection followed by tuple set reduction, or as a tuple-wise null-extended union of the usual projection.

Definition 7.18: The *null-extended projection* of relation r on attributes X is

$$\pi_X^{e'}(r) = \widehat{\{t \mid t \in \pi_X^e(r)\}} = \bigcup_{t \in r[X]}^{e'}(t)$$

Proposition 7.14: Null-extended projection is a precise generalization of standard projection with respect to unnesting and PNF possibility function *POSS**.

Proof: Since the only difference between null-extended projection and extended projection is removal of subsumed tuples, the proof mirrors the proof for null-extended union (Proposition 7.2). \square

7.3 Dependencies in a Database with Null Values

A key assumption made in this chapter has been the requirement of partitioned normal form. In the definition of PNF, we assume that certain multivalued dependencies must hold in a 1NF relation before it can be legally nested into a particular form. Furthermore, multivalued dependencies imply functional dependencies in the nested relation. Therefore, it is important to determine what effect the addition of null values will have on these dependencies.

In this section we will discuss the previous work on extending depen-

r

A	B	C	D
a_1	dne	c_1	d_1
a_1	dne	c_2	d_2
a_2	b	c_1	d_1
a_2	b	c_2	d_2
a_2	b	c_2	d_1
a_2	b	c_1	d_2

Figure 7-13. Relation satisfying $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$, but not $A \twoheadrightarrow C$ when dne nulls are not equated.

dependencies to deal with nulls, providing some new clarifying information. We will examine how these dependencies interact with the non-existent, unknown, and no-information interpretation of nulls.

7.3.1 Non-existent Nulls

In [Lie3], a sound and complete axiomatization for functional and multivalued dependencies are given for a relational model in which dne nulls are allowed. In this model, dne nulls are not considered equal to each other. Notably missing from the inference rules for both FDs and MVDs is the transitivity rule. The problem occurs when dne nulls appear in the attribute that implements the transitivity, as the application of the FD and MVD rules is denied when null values are present on the left hand side of the rule.

An example for MVDs is a relation r on scheme $R = (A, B, C, D)$ where $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$ hold, but $A \twoheadrightarrow C$ does not hold (Figure 7-13).

We assume a model of a relation in which tuples or fragments of tuples represent fundamental relationships in the world being modeled. Each set of attributes that is involved in one of these fundamental relationships is called an *object* [FMU]. On examining the first two tuples in relation r , it must be true that there is an object involving attributes A , C , and D , and no subset of them. Otherwise, we would have to add two tuples matching the first two tuples in r but with the C and D values swapped. However, on examining the last four tuples, where dne nulls do not occur, there are independent AC and AD objects. If we accept this, then we must accept the fact that there are two different semantics for tuples in r . If the value of B is dne then an ACD association must exist, and if the value is not dne then independent AC and AD associations must exist, in addition to associations involving B . We do not believe this is a plausible way to interpret a relation.

The solution is to equate dne nulls from the same domain. Then, in a database with only dne nulls added, the definitions of FD and MVD remain identical to the standard ones and the same axiomatization is valid. This is intuitively pleasing as well, since a dne null cannot be replaced by another value. In fact, it indicates that we know that no other domain value is valid.

Non-existent nulls also require a more complicated test when tuples are inserted into a relation. In addition to the usual tests to see that given dependencies are not violated, we must ensure the exclusivity of the dne null

in each object in which it appears. For example, let us attempt to add the tuple $\langle a_3, b, dne, d_3 \rangle$ to relation r above. This insertion should be denied since it is inconsistent that b is related to c_1 and c_2 and also that b is related to no C value. This new integrity constraint is embodied in the following rule.

Exclusivity Rule for dne Nulls: Let r be a relation with objects O . For each $O \in \mathcal{O}$, in $\pi_O(r)$ there do not exist two tuples t_1 and t_2 where $t_1[A] = dne$, $t_1[A] \neq t_2[A]$, and $t_1[O - A] = t_2[O - A]$, for any $A \in O$.

7.3.2 Unknown Nulls

The effect of **unk** nulls on functional dependencies has been adequately covered in [Vas1]. The definition of an FD must be modified so that **unk** nulls are not equivalent. This must be the case since we have no way of knowing whether two **unk** nulls will turn out to be the same or different values. The same logic holds for MVDs. However, unlike the assumptions made by [Lie2, Lie3] for **dne** nulls, even though we cannot apply an FD to adjust values or an MVD to add tuples when there are **unk** nulls on the left hand side of the dependency, we still have the usual axiomatization for FDs and MVDs. In proof, suppose we have a relation that satisfies some given dependencies, but not some dependency which follows from the usual axiomatization. An example is relation r in Figure 7-13, with **unk** nulls replacing the **dne** nulls. Since **unk** nulls are placeholders for actual facts about the world, the dependencies with which we have constrained

the world are not altered by the presence of these nulls. Therefore, dependencies which follow from the given dependencies in a world without null values must still hold in a world with nulls. Thus, a relation such as r with **unk** nulls, must not be a complete or accurate representation of the world, since for any relation r , every relation in $POSS(r)$ must satisfy all FDs and MVDs which can be derived from the given dependencies.

7.9.3 No-information Nulls

The only published work dealing with dependencies and the no-information interpretation of nulls is an axiomatization of FDs by [AM]. As in previous approaches, they redefine the FD so that it is applicable only when non-null values are present. Therefore, they conclude the same results as [Lie3], about the lack of transitivity in this model. Based on the lattice developed in section 7.1, we know that an **ni** null will eventually be replaced by either an **unk** null or a **dne** null when we find out whether or not a value actually exists. Hence, given a relation r with **ni** nulls, in any relation in $POSS(r)$ all **ni** nulls will be replaced by actual values or by **dne**. As discussed earlier in this section, in these cases, there is no valid reason not to retain the same axiomatization for FDs and MVDs as for relations without nulls, and to do so would possibly eliminate important dependencies for use in database design and normalization. Thus, we repeat an earlier statement, that the definitions of FD and MVD need not be changed as long as the convention that two values from the same extended

domain are equal if they are the same value and neither one is *ni* or *unk*.

7.3.4 Join Dependency

At first glance, there doesn't seem to be any good way to define the join dependency on relations with nulls. Consider the tuple $\langle a, \text{ni}, c \rangle$ defined on scheme $R = (A, B, C)$. Normally any one tuple relation satisfies any join dependency since any projections of the tuple will obviously join to form the original tuple. However, with the given tuple, the join dependency $*(AB, BC)$ does not hold since the projections will not join on *ni*. However, the MVD which follows from this join dependency, $B \twoheadrightarrow A$, does hold by default. What we need is a "default" for the join dependency when *ni* or *unk* nulls are present in the join attributes. We have decided that, in general, *ni* and *unk* nulls should not be equated with each other. However, each null does stand for one and only one value (actual or *dne*), and so if a null is transported to more than one place we should identify them to be the same. Therefore, we *mark* *ni* and *unk* nulls before applying the test for satisfying the join dependency, doing so by equating identically marked nulls. We now have an appropriate definition for a join dependency in our framework and we can use the existing theory for deriving MVDs from valid join dependencies.

Chapter 8

The SQL/NF Query Language

In Chapter 4, we defined a formal predicate-calculus-based language for dealing with \neg 1NF relations. That language defines a minimal degree of power that we expect from any language designed to operate on nested relations. For real-world users of a database system, however, a terse predicate-calculus language is too difficult to use. These considerations have led to the definition of several “syntactically-sugared” query languages such as SQL, Query-By-Example, and QUEL. In this chapter, we extend one of the most widely-used of these languages, SQL [C+], to operate on a database of nested relations. Most of our extensions pertain to the data manipulation part of SQL, although we extend also the SQL data definition language to permit the definition of \neg 1NF databases. In defining SQL/NF, an important goal was to retain the “spirit” of the existing SQL language so as to reduce the effort required on the part of existing SQL users to learn SQL/NF. Roughly speaking, wherever a constant or scalar-valued variable may appear in SQL, a relation or expression evaluating to a relation may appear in SQL/NF. We introduce new commands to transform a relation to an equivalent nested one (the *nest* operation) and to transform a nested relation into a less-nested one (the *unnest* operation). We shall assume that the reader is familiar with standard SQL as described in [C+] or in most standard database texts.

Although we retain most constructs of the SQL language, we would be remiss if we did not make some obvious improvements in the SQL language. Some of these changes are due to the availability of nested relations. For instance, difficult SQL queries involving GROUP BY and HAVING can be eliminated in favor of rather straightforward queries on properly structured \neg 1NF relations. Other changes are simply to correct some mistakes made in the design of SQL. In Date's critique of the SQL language [Dat1], a good case is made for requiring certain modifications to the SQL language. One of the driving forces behind the critique is the language design maxim, the *principle of orthogonality*. This principle requires separate treatment for distinct concepts, and similar treatment for similar concepts [Dat2]. The following definition of the SQL/NF language takes into account this principle, directly incorporating some of the modifications proposed in [Dat1]. We also follow where possible the proposed standard relational database language [X3H2], which already incorporates some the changes suggested here, although relying on a 1NF database model.

The remainder of this chapter is organized as follows. Sections 8.1–8.3 contain our definition and description of the SQL/NF language. We define the query facilities, the data manipulation language and the data definition language. Host language support is beyond the scope of this dissertation. Section 8.4 compares our language with some previous attempts at defining a high-level query language for \neg 1NF models. We also provide a BNF definition of

our language in Appendix B.

8.1 Query Facilities

In SQL, a basic query conforms to the structure

```
SELECT attribute-list
FROM   relation-list
WHERE  predicate
```

This SFW-expression can be conceptually executed by forming the cartesian product of all relations in the *relation-list*, choosing only tuples in this product that satisfy the *predicate*, and then choosing only those attributes in the *attribute-list*. If no qualification of tuples is needed then the WHERE clause can be omitted. If all attributes of the relations in the *from-list* are desired then SQL allows the use of “*” instead of actually specifying all attributes in the *attribute-list*. In the proposed standard relational database language (hereafter called RDL), the keyword ALL is used instead of “*” [X3H2].

The obvious way to access the entire contents of a relation would be to simply state the relation name. However, in SQL, we have to use

```
SELECT *
FROM   relation-name
```

In SQL/NF, we allow free substitution of *relation-name* for “SELECT * FROM *relation-name*” and adopt the RDL change substituting ALL for “*”. Consider the 1NF database in Figure 8-1. The department (*Dept*) relation has three attributes, department number (*dno*), department name (*dname*), and location

dno	dname	loc
10	Manufacturing	Austin
20	Personnel	Dallas
30	Retail	Austin
⋮	⋮	⋮

eno	ename	dno	sal
13	Smith	10	20000
33	Jones	30	14000
34	Adams	10	15000
48	Miller	10	40000
⋮	⋮	⋮	⋮

Figure 8-1. A sample 1NF database.

(*loc*). The employee (*Emp*) relation has four attributes, employee number (*eno*), employee name (*ename*), department number of department in which employee works (*dno*), and salary (*sal*). The query to get employee data for employees in department 10 is

```
SELECT ALL
FROM   Emp
WHERE  dno = 10
```

or using our simplified notation,

```
Emp WHERE dno = 10
```

To get departments which have at least one employee, the query is

```
SELECT ALL
FROM   Dept
WHERE  EXISTS (SELECT ALL
                FROM   Emp
                WHERE  Dept.dno = Emp.dno)
```

or, in SQL/NF,

```
Dept WHERE EXISTS (Emp WHERE Dept.dno = Emp.dno)
```

This last query easily paraphrases as: Get department tuples where there exists

employee tuples where the department numbers are the same. The same can not be said about the strict SQL version. In fact it is not clear why we are selecting any attributes at all in the EXISTS subquery, since our goal is not to actually extract any information from the employee relation but rather to test for its existence.

8.1.1 Nested Expressions

A language should provide, for each class of object it supports, a general, recursively defined syntax for expressions that exploits to the full any closure properties the object class may possess [Dat1; 12].

The primary objects a relational database language supports are scalar (atomic) values and relations. In 1NF databases, each relation is comprised strictly of scalar values. In \neg 1NF databases, each relation may be comprised of other relations as well as scalar values. The principle of orthogonality has been usefully employed in defining the \neg 1NF data structure. Wherever a scalar value could occur in a 1NF relation, a relation can now occur. This simple transformation is also employed in our definition of the data sublanguage. SQL has the *closure* property where the result of any query on one or more relations is itself a relation. The principle of orthogonality suggests that we should allow an SFW-expression wherever a relation name could exist. In SQL this means allowing SFW-expressions in the FROM clause.

The first use of such a modification is the building of incremental

queries. Using the database of Figure 8-1, consider the query: Get names of employees who work in the shipping department. The first step a user may recognize is the the need to join the Emp and Dept relations on dno. So he forms the query

```
SELECT ALL
FROM   Emp, Dept
WHERE  Emp.dno = Dept.dno
```

Then from this relation get names of employees in the shipping department producing

```
SELECT ename
FROM    (SELECT ALL
        FROM   Emp, Dept
        WHERE  Emp.dno = Dept.dno)
WHERE   dname = "Shipping"
```

An SQL/NF-level query optimizer could then translate this query into the equivalent query

```
SELECT ename
FROM    Emp, Dept
WHERE   Emp.dno = Dept.dno
AND     dname = "Shipping"
```

A more useful example involving nested expressions in the FROM clause involves the UNION operator. UNION is an infix operator in SQL and is used in the form

SFW-expression UNION *SFW-expression*

Note that, in SQL, one must use SFW-expression's with UNION and not relations. To illustrate, suppose we have two employee relations (in the form of

Figure 8-1). The *Emp-exec* relation contains executive level employees and the *Emp-other* relation contains all other employees. If we want to get all employees, we write the SQL query

```
SELECT *
FROM   Emp-exec
UNION
SELECT *
FROM   Emp-other
```

In SQL/NF, we can write the simpler query

```
Emp-exec UNION Emp-other
```

Now, if we modify our query so that we get all employees who make more than \$35,000, then we must add a *WHERE* clause to each *SFW*-expression in the SQL query.

```
SELECT *
FROM   Emp-exec
WHERE  sal > 35000
UNION
SELECT *
FROM   Emp-other
WHERE  sal > 35000
```

Using SQL/NF, we can form the union first, place that expression in the *FROM* clause, and qualify all tuples with one *WHERE* clause.

```
SELECT ALL
FROM   (Emp-exec UNION Emp-other)
WHERE  sal > 35000
```

Nested expressions are even more applicable when using a \neg 1NF database. Since attributes may now be relation valued, relation names may occur

Company

dno	dname	loc	Emps		
			eno	ename	sal
10	Manufacturing	Austin	13	Smith	20000
			34	Adams	35000
			48	Miller	40000
20	Personnel	Dallas			
30	Retail	Austin	33	Jones	14000
⋮	⋮	⋮		⋮	

Figure 8-2. A sample \neg 1NF database.

in the **SELECT** clause of a query, and so, under the principle of orthogonality, we allow *SFW*-expressions in the **SELECT** clause. For the following examples we will use the \neg 1NF database in Figure 8-2, in which we have combined the data of the *Dept* and *Emp* relations used in Figure 8-1. The *company* (*Company*) relation has four attributes, department number (*dno*), department name (*dname*), location (*loc*), and employees (*Emps*). Each *Emps* relation has three attributes, employee number (*eno*), employee name (*ename*), and salary (*sal*). Recall that for notational simplicity, we eliminate the set braces which normally would occur around each *Emps* relation in Figure 8-2. Consider a query to get department names and the employees in each department making more than \$35,000. First consider getting all employees. The query is

```
SELECT dname, Emps
FROM Company
```

Since *Emps* is a relation we could, equivalently write

```

SELECT dname, ( SELECT ALL
                  FROM Emps)
FROM Company

```

Now to limit employees to those making more than \$35,000, it is a simple matter of adding a WHERE clause to the nested SFW-expression, giving

```

SELECT dname, ( SELECT ALL
                  FROM Emps
                  WHERE sal > 35000)
FROM Company

```

or

```

SELECT dname, (Emps WHERE sal > 35000)
FROM Company

```

Note that a SFW-expression may produce an empty relation. In the last query the "Personnel" tuple will have an empty Emps relation since it was empty to begin with, and the "Retail" tuple will have an empty Emps relation since none of its employees satisfy the "sal > 35000" predicate. To eliminate tuples with empty *Emps* relations in the result we can write

```

SELECT dname, (Emps WHERE sal > 35000)
FROM Company
WHERE EXISTS(Emps WHERE sal > 35000)

```

Later, we introduce a technique for referencing the new *Emps* relation the first time it is mentioned, avoiding the duplicate specification of the nested query.

Not only can we select specific tuples from a nested relation we can also select specific attributes. To illustrate, consider the query to get department names and locations and employee names and salaries. We write


```

SELECT dname, loc, (SELECT ename, sal
                    FROM Emps)
FROM Company

```

Combining the above techniques, we can write the following query. Get department names and locations, and employee names and salaries where the location is 'Austin' and the employee salary is more than \$35,000.

```

SELECT dname, loc, ( SELECT ename, sal
                    FROM Emps
                    WHERE sal > 35000)
FROM Company
WHERE loc = 'Austin'

```

8.1.2 Functions

In SQL, the argument to a function such as SUM is a column of scalar values and the result is a single scalar value; hence, orthogonality dictates that (a) any column-expression should be permitted as the argument, and (b) the function-reference should be permitted in any context in which a scalar can appear. However, (a) the argument is in fact specified in a most unorthodox manner, which means in turn that (b) function references can actually appear only in a very small set of special-case situations [Dat1; 20].

Date's arguments are even more valid when we assume a \neg 1NF model. Here we have built-in sets of values in the form of nested relations and it would make more sense to apply functions to relations rather than artificially applying them to attributes. Then, by the principle of orthogonality, we should be able to apply functions to any expression that evaluates to a relation.

Consider first the 1NF database of Figure 8-1, and a query to find the

total amount made by all employees. In SQL, we would write

```
SELECT SUM(sal)
FROM Emp
```

The argument to SUM is actually the entire *sal* column of *Emp*, whereas a reference to *sal* in a WHERE clause (e.g., *sal* > 5000), is referring to individual *sal* values. Therefore, we adopt Date's suggestion to apply functions to their actual argument. Thus, our query becomes

```
SUM(SELECT sal
FROM Emp)
```

Another example is the query which gets all departments that employ more than 10 people:

```
SELECT dno
FROM Dept
WHERE COUNT(SELECT *
FROM Emp
WHERE Dept.dno = Emp.dno)
> 10
```

or using our simplified notation which substitutes "Emp" for "SELECT * FROM Emp":

```
SELECT dno
FROM Dept
WHERE COUNT(Emp WHERE Dept.dno = Emp.dno) > 10
```

In SQL, the latter query would usually be formulated using GROUP BY and HAVING.

```

SELECT dno
FROM   Dept
WHERE  dno IN
      (SELECT  dno
       FROM    Emp
       GROUP BY dno
       HAVING  COUNT(DISTINCT eno) > 10)

```

GROUP BY introduces a new structure into the relational model: partitioned relations. The only attributes which can be selected from a partitioned relation are the "group by" attributes, i.e., those that have the same value for each partition, and single-valued functions of any attribute. Normally, a function operates on the entire relation, but when a relation is partitioned, the function is applied separately to each partition. Thus, we have a new structure which, incidentally, is not in 1NF, with new rules for the execution of SFW-expressions, and a new HAVING clause to test predicates on partitions.

In some cases GROUP BY and HAVING are not necessary. An example of this is when the values of the functions are not being retrieved in a SELECT clause. For example, a legal SQL query to do the last query is

```

SELECT dno
FROM   Dept
WHERE  10 < (SELECT COUNT(*)
            FROM   Emp
            WHERE  Emp.dno = Dept.dno)

```

Furthermore, if we allow nested queries in the SELECT clause then GROUP BY and HAVING are totally unnecessary. For example, to retrieve the counts of employees for each department we could write

```
SELECT dno, COUNT(Emp WHERE Emp.dno = Dept.dno)
FROM Dept
```

Now let us consider the \rightarrow 1NF database in Figure 8-2. Since employees have already been “grouped by” department, our queries are easier to formulate.

To get the employee counts, we write

```
SELECT dno, COUNT(Emps)
FROM Company
```

To get departments where the employee count is more than 10, we write

```
SELECT dno
FROM Company
WHERE COUNT(Emps) > 10
```

By structuring relations appropriately, we can turn any GROUP BY/HAVING query into a straightforward SFW-expression. Since these types of queries are some of the hardest to formulate in SQL, and operate under a different set of rules from standard SQL queries, their elimination is a major advantage of the \rightarrow 1NF model.

A further advantage of using relations or nested expressions as input to functions is the ability to use multi-attribute relations and have the function apply to several attributes simultaneously. For example, suppose we have a *Sales* relation with employee number (*eno*) and 12 sales attributes (*Jan-sales*, *Feb-sales*, ..., *Dec-sales*) showing total sales for each month of the year for the employee. Then to get the total of all sales in each month we can write

```
SUM(SELECT Jan-sales, Feb-sales, Mar-sales, Apr-sales, May-sales, Jun-sales,  
      Jul-sales, Aug-sales, Sep-sales, Oct-sales, Nov-sales, Dec-sales  
FROM   Sales)
```

The SUM function is applied to each column of the argument relation. In general, a *column* function, (SUM, AVG, MAX, MIN), reduces a relation to a single tuple with the same number of attributes, by applying the function to each column of the relation. A *table* function (COUNT), reduces a relation to a single tuple with one attribute. Thus, the result of applying a function is always a single tuple relation.

8.1.3 Null Values and Operations Dealing with Nulls

One question that usually arises when dealing with functions concerns the presence of null values. SQL makes the decision to ignore null values in all functions except COUNT. An unfortunate consequence of this is that the equality of $AVG(Rel) * COUNT(Rel)$ and $SUM(Rel)$ may be violated. We believe that nulls should not be ignored in any function, rather they should, when appropriate, produce an error. This forces the user to remove the nulls before applying the function and also prevents him from believing he has received a precise answer to a query which is, in fact, based on imprecise data.

A thorough treatment of nulls for $\neg 1NF$ databases was found in Chapter 7. We saw that one of the functions which is usually required when dealing with null values is a method for eliminating *subsumed* tuples in a relation.

In SQL/NF we allow a single atomic null-value, denoted NULL. A tuple t is *subsumed* by tuple q if t 's non-null attributes have the same values as the corresponding attributes in q . For example, the tuple $t = \langle \text{Smith}, \text{NULL}, \text{NULL} \rangle$ is subsumed by $\langle \text{Smith}, 10, \text{NULL} \rangle$ and by $\langle \text{Smith}, 20, 15000 \rangle$, but not by $\langle \text{Jones}, \text{NULL}, 15000 \rangle$. Subsumed tuples are like duplicate tuples in that they do not provide any more information than some other tuple in the relation. When nested relations are attributes the definition is applied recursively, so that relation r subsumes relation s , if every tuple in s is subsumed by some tuple in r .

Although SQL eliminates duplicate tuples via the SELECT DISTINCT construct, it does not eliminate subsumed tuples, even though null values are allowed. Therefore, we introduce the SUBSUME function to eliminate subsumed tuples from a relation. Note, that SUBSUME also removes duplicate tuples, since by definition if $t = q$ then t subsumes q and q subsumes t . We also use our standard notation for applying a function for the syntax of DISTINCT and SUBSUME.

To eliminate duplicates from the Company relation we use

DISTINCT(Company)

To get department names and employees names and salaries, eliminating subsumed employee tuples, we use

```
SELECT dname, SUBSUME(SELECT ename, sal
                       FROM Emps)
FROM Company
```

If we want to eliminate duplicates before counting the number of tuples in the Company relation, our query is

```
COUNT(DISTINCT(Company))
```

Another important operation which becomes available when null values are supported is the *outer join*. The outer join is similar to a traditional join, except that tuples which normally would not participate in the join are added to the result. Null values are used for the attributes not in the relation. In [Dat3], a proposal is made for supporting the outer join operation with a PRESERVE clause. All tuples of the relations specified in the PRESERVE clause are included in the resulting relation even if they do not satisfy the predicates of the WHERE clause. The attributes of the resulting relation which are not in the "preserved" relation are set to NULL for those tuples which did not satisfy the WHERE clause.

For example, to join the *Dept* and *Emp* relations in our 1NF database, without losing the department data for departments that do not have any employees, we would use the PRESERVE clause as follows.

```
SELECT *
FROM Dept, Emp
WHERE Dept.dno = Emp.dno
PRESERVE Dept
```

Finally, a clarification of the relationship between empty relations and null values is in order. For reasons discussed in Chapter 7, we note that the empty relation is equivalent to any relation in which all attributes of all tuples have null values for the atomic attributes and, recursively, empty relations for the nested relations. Under subsumption, all of these relations are equivalent to a single tuple relation, where the value of each attribute is null or empty. Since we have a single type of null in SQL/NF, we assume the most general interpretation, that is, the *no-information* interpretation. This means we do not know whether or not an actual value exists which could replace this null. Since empty relations are equivalent to a relation with null-tuples, we assign the no-information interpretation to empty relations as well.

8.1.4 Miscellaneous Features

8.1.4.1 Unnesting after a Function

When a column or table function is applied to a nested relation it doesn't make sense to retain the relation structure for a single tuple. Therefore, our functions will cause the relation in which it occurs to be unnested one level. For example, instead of the result of our query to get department numbers and the number of employees in each department having tuples $\{ \langle 10, \{3\} \rangle, \langle 20, \{0\} \rangle, \langle 30, \{1\} \rangle, \dots \}$, we would have $\{ \langle 10, 3 \rangle, \langle 20, 0 \rangle, \langle 30, 1 \rangle, \dots \}$. This feature also allows easier application of multiple functions. For instance, to get

the total number of employees in the company from our \rightarrow 1NF database we would write

```
SUM(SELECT COUNT(Emps)
      FROM Company)
```

Without the COUNT function unnesting its sets the SUM function would get sets of counts as arguments and would not work properly.

8.1.4.2 Attribute Lists

Sometimes it is easier to list the attributes you do not want to deal with. For this, SQL/NF allows the construct "ALL BUT *attribute-list*". Recall a previous example in which we were interested in getting the total sales in each month from a *Sales* relation with employee number and 12 sales attributes, one for each month. In that query we had to list all 12 sales attributes, when it would be much easier to list the one attribute we were not interested in, *eno*. Our query then becomes

```
SUM(SELECT ALL BUT eno
      FROM Sales)
```

8.1.4.3 Don't Care Values

When comparing constant values with attributes values in a "don't care" value is useful for making wild card comparisons. Our "don't care" value is the question mark (?). To illustrate its use, consider the query to get *Company* tuples where one of the employees has name "Smith" and salary \$20,000. One way to write this query is to look for an employee tuple with "ename = 'Smith',

"sal = 20000", and any value for *eno*. We can use our "don't care" value as follows

Company WHERE <?, "Smith", 20000> IN Emps

This is certainly more straightforward than the alternative

Company WHERE EXISTS
 (Emps WHERE ename = "Smith"
 AND sal = 20000)

Various other text matching facilities could be incorporated. RDL has a proposed text matching facility based on the SQL "LIKE" predicate, and much of Schek's work (cf. [Sch1]) has been involved with text retrieval in a database system.

8.1.5 Data and Relation Restructuring Operations

Two operations, NEST and UNNEST are provided for restructuring relations into either more or less nested forms. One operation, ORDER, rearranges the tuples of a relation.

The restructuring operations correspond to the nest and unnest operators of the π -1NF relational algebra. The syntax of these operators is

NEST (<i>query</i>) ON <i>attribute-list</i> [AS <i>name</i>]	UNNEST (<i>query</i>) ON <i>attribute-list</i>
---	---

In the following, let *Rel* be the relation formed by (*query*). The NEST operation partitions *Rel* on the attributes not specified in the *attribute-list*. For

each partition, a new tuple is created with the values of the attributes in the *attribute-list* collected into a new nested relation. The nested relation is given an optional *name* but, if not named, it cannot be referenced any place else in the query. Note that if any nested relation formed consists solely of tuples in which every attribute has a value which is null or the empty relation, then the value of this nested relation is the empty relation.

Let us go through a step by step building of a query to convert the 1NF database of Figure 8-1 to the 3NF database of Figure 8-2.

First we will nest the employee relation by collecting *eno*, *ename*, and *sal* into a nested relation called *Emps*.

```

NEST  Emp
ON    eno, ename, sal AS Emps

```

Next, we join this relation with the *Dept* relation on *dno* and eliminate one of the duplicate *dno* columns.

```

SELECT ALL BUT Emp.dno
FROM   Dept, (NEST Emp
              ON   eno, ename, sal AS Emps)
WHERE  Dept.dno = Emp.dno

```

This query produces all tuples in the *Company* relation where departments have employees. If we want to also include the departments which do not have employees, assigning a null tuple to the nested *Emps* relation, we need to *preserve* the *Dept* relation. The final query is

```

SELECT ALL BUT Emp.dno
FROM   Dept, (NEST Emp
              ON   eno, ename, sal AS Emps)
WHERE  Dept.dno = Emp.dno
PRESERVE Dept

```

The UNNEST operation creates several tuples for each tuple in *Rel*, by concatenating the attributes not specified in the *attribute-list* with a tuple from each of the attributes that is specified. The attributes of the unnested relations now become attributes of *Rel*. Note that an empty relation unnests to a single tuple with null values for each atomic attribute and an empty relation for each nested relation.

To unnest the *Company* relation we write

```

UNNEST (Company)
ON     Emps

```

To convert the \neg 1NF database to the 1NF database we issue a query for each relation. To get the *Emp* relation we write

```

SELECT eno, ename, dno, sal
FROM   (UNNEST (Company)
        ON     Emps)
WHERE  eno IS NOT NULL

```

and, to get the *Dept* relation we write

```

SELECT dno, dname, loc
FROM   Company

```

In SQL, the ORDER BY clause is added to a query if tuples are to be sorted in some particular order before being output to the user. We retain this

function, but modify its syntax to match the other functions in our language.

The new syntax is

```
ORDER (query)  
  BY name [ASC | DESC] {, name [ASC | DESC]}
```

To sort the *Company* relation into ascending order by location, we write

```
ORDER Company  
  BY loc ASC
```

8.1.6 Name Inheritance and Aliasing

The attributes of the relations formed in the FROM clause of a SFW-expression may be used in several places in a query. They may be referenced directly in (1) the SELECT clause, (2) the FROM clause of a nested SFW-expression, or (3) the WHERE clause. Attributes may be referenced also in the WHERE clause of any nested SFW-expression. A problem occurs when attribute names are not unique. This can be due to the need to use multiple copies of a relation in a single query or to the presence of identical names in different relations, or nested relations.

In the first case, when we need to use multiple copies of a relation, the solution is to introduce *reference names* for the relations. For example, the query to get all pairs of department names that exist at the same location requires reference name for the *Company* relation. A reference name is specified by including the key word AS and the new name.

```

SELECT First.dname, Second.dname
FROM   Company AS First, Company AS Second
WHERE  First.loc = Second.loc AND First.dno < Second.dno

```

If necessary, or desired, reference names can also be used for nested SFW-expressions and for attribute names. If a single relation *X* is specified in the *FROM* clause of a SFW-expression the name of the resulting relation defaults to *X*. However, if there is more than one relation in the *FROM* clause, then there is no default, and a reference name is required if the resulting relation is going to be referenced elsewhere in the query. Consider the last query to get pairs of department names. Let us use the result of this query to get all triples of department names at the same location, and rename the attributes to *dname1*, *dname2*, and *dname3*. We will use the reference name *Pairs* for the last query and use *Pairs2* for a new reference name for *Pairs*.

```

SELECT Pairs.First.dname AS dname1, Pairs.Second.dname AS dname2,
       Pairs2.Second.dname AS dname3
FROM   (SELECT First.dname, Second.dname
        FROM   Company AS First, Company AS Second
        WHERE  First.loc=Second.loc AND First.dno<Second.dno) AS Pairs,
       Pairs AS Pairs2
WHERE  dname2 = Pairs2.First.dname

```

Note how reference names are cascaded when necessary to distinguish attribute names. Similarly, if it is necessary to distinguish identical names that occur at different nesting levels of a relation then each *name* attribute can be prefixed by the relation name of the nested relation in which it occurs. In addition, unnesting a relation via the *UNNEST* operator may require that the

unnested relation's name be attached to any names which would otherwise be identical in the resulting relation.

Reference names are useful for simplifying queries in which a nested query expression is used in several places in the query. Recall from section 8.1.1 the query to get department names and employees making more than \$35,000, eliminating departments with no employees meeting the salary requirement.

Our solution then was

```
SELECT dname, (Emps WHERE sal > 35000)
FROM   Company
WHERE  EXISTS(Emps WHERE sal > 35000)
```

By using a reference name for the nested query on *Emps*, the duplication can be eliminated, as follows:

```
SELECT dname, (Emps WHERE sal > 35000) AS Emps-rich
FROM   Company
WHERE  EXISTS(Emps-rich)
```

8.2 Data Manipulation Language

In this section we discuss commands to store, modify, and erase data from relations in the database. These commands can be thought of as functions which transform relations into other relations by adding, changing, or deleting data from them. Just as an SFW-expression produces relations from relations, the DML commands perform similarly with the additional effect that the new relations replace the old relations in the database. Thinking of DML commands as functions is critical if we want to apply them to \neg 1NF relations. In a \neg 1NF

model we will need to manipulate nested relations as easily as we manipulate traditional relations.

To get the feel for our syntax (adapted from the RDL standard [X3H2]), let us start with some examples on the 1NF database of Figure 8-1. The **STORE** statement can be used to add user specified tuples to a relation or to add tuples retrieved via a query specification to a relation. To add two new departments to the *Dept* relation, we write

```
STORE Dept
VALUES <50, Training, Waco>
      <60, Sales, Austin>
```

Suppose we had a relation *New-Dept* with two attributes, *deptno* and *deptname*, which contained information on some new departments. If we want to store this data in the *Dept* relation we write

```
STORE Dept(dno, dname)
SELECT ALL
FROM New-Dept
```

For each of the *New-Dept* tuples stored in *Dept*, the *loc* attribute will be set to the default value defined for *loc* in the schema definition, or NULL if no default value was specified. In general, an arbitrary SFW-expression can be used in the **STORE** command to specify the tuples to be stored. Of course, the relation created must be compatible (number of attributes and domain types) with the relation being stored into.

The **MODIFY** command is used to replace values with others in the

database. Suppose we want to give every employee in the *Emp* relation a 10% raise. We would write

```
MODIFY Emp
  SET sal = sal * 1.1
```

If we want to limit the raise to those employees in department 10, we add a *WHERE* clause to the query as follows:

```
MODIFY Emp
  SET sal = sal * 1.1
  WHERE dno = 10
```

In general, we can specify more than one replacement in the *SET* clause, and qualify the tuples to be modified via an arbitrary predicate in the *WHERE* clause.

The *ERASE* command is used to delete tuples from relations. An optional *WHERE* clause is used to identify the tuples to be deleted. Let us delete all departments with no employees.

```
ERASE Dept
  WHERE NOT EXISTS (Emp WHERE Dept.dno = Emp.dno)
```

All three DML commands operate by first computing all changes, and then making all changes to the relation in one atomic action. This way the relation being changed may be referenced in a nested SFW-expression without fear of it changing while the command is being executed. For example, suppose we want to delete all employees whose salary is greater than the current average salary. The appropriate *ERASE* command is

```
ERASE Emp
  WHERE sal > AVG(SELECT sal FROM Emp)
```

If, instead of the above rule, we recalculated the average salary as we checked and perhaps deleted each tuple in *Emp*, it is possible to wind up deleting all tuples in *Emp*!

Now let us focus on the particular problem that \neg 1NF relations pose for our DML commands. We need a way of performing the three DML commands on individual nested relations. No matter which operation we perform on a nested relation, we are changing only the relation in which the updated relation is nested. Therefore, all changes to nested relations are done with a MODIFY command on the database relation. For the next set of examples we use the \neg 1NF database of Figure 8-2. Suppose we want to insert a new employee, $\langle 32, \text{Samuels}, 49000 \rangle$, working in department 10. An outline of the required command is

```
MODIFY Company
  SET Emps = X
  WHERE dno = 10
```

Since *Emps* is a nested relation, what should we use for *X* in this operation? Since we allow any atomic-valued expression to be used when the attribute being changed is atomic, we allow any relation valued expression to be used when the attribute being changed is a relation. Thus, one legitimate solution is to replace *X* with

```
(Emps UNION  $\langle 32, \text{Samuels}, 49000 \rangle$ )
```

Another, more general solution is to replace *X* with a “nested” STORE command

on the *Emps* relation. The total query is then

```

MODIFY Company
  SET Emps = (STORE Emps
              VALUES <32, Samuels, 49000>)
WHERE dno = 10

```

In general, we can use UNION instead of STORE and DIFFERENCE instead of ERASE, however, there is usually not a good way to simulate MODIFY using a query expression. The command to give each employee in department 10 that makes more than \$30,000, a 10% raise, is written

```

MODIFY Company
  SET Emps = (MODIFY Emps
              SET sal = sal * 1.1
              WHERE sal > 30000)
WHERE dno = 10

```

The alternative query expression for the nested MODIFY is the much more complex expression:

```

SELECT eno, ename, sal * 1.1
FROM   Emps
WHERE  sal > 30000
UNION
Emps WHERE sal <= 30000

```

In summary, when tuples are to be stored, modified, or erased from a nested relation, either a query expression can be constructed to perform the modification or the appropriate DML command can be used. In either case, any operation on a nested relation is done within a MODIFY command on the relation containing the nested relation.

8.3 The SQL/NF Data-Definition Language

In standard SQL, it is possible to define relations using the `CREATE TABLE` command, and views using the `DEFINE VIEW` command. As part of the `CREATE TABLE` command, the user specifies the attribute names and the domain (e.g., integer, character) to be associated with each attribute of the relation. In the proposed RDL standard, base tables and views are defined in a `SCHEMA` command, which includes a `TABLE` command for each base table being defined, and a `VIEW` command for each view being defined. In addition to specifying the attribute names and their domains, a variety of integrity constraints can also be specified (`UNIQUE`, `NOT NULL`, `REFERENCES ...`, `CHECK ...`)¹.

We shall adopt the RDL framework for the SQL/NF data-definition language. However, we shall need to make appropriate modifications to allow for definition of \neg 1NF relations. Let us first show the definitions for the 1NF database in Figure 8-1.

```

SCHEMA
  TABLE Dept
    ITEM dno INTEGER UNIQUE NOT NULL
    ITEM dname CHARACTER 10
    ITEM loc CHARACTER 10
  TABLE Emp
    ITEM eno INTEGER UNIQUE NOT NULL
    ITEM ename CHARACTER 10
    ITEM dno INTEGER REFERENCES Dept.dno
    ITEM sal REAL

```

¹ See [X3H2] for details on these constraints.

Each ITEM command defines a column in the relation. The UNIQUE constraint specifies that no duplicates are allowed for the attribute (thus forming a *key* for the relation). The NOT NULL constraint specifies that no null values are allowed for the attribute, and the REFERENCES constraint disallows any value for the attribute that is not a value in the referenced column. Note that these constraints are also allowed as separate clauses in a TABLE definition. This is especially needed when two or more attributes are to be key for a relation and their combination must be specified as UNIQUE (see the Appendix for syntax).

Following the principle of orthogonality, in order to define \rightarrow 1NF relations, we must allow TABLE definitions wherever an atomic-valued specification could occur before. The definitions for the \rightarrow 1NF database of Figure 8-2 are:

```

SCHEMA
  TABLE Company
    ITEM dno INTEGER UNIQUE NOT NULL
    ITEM dname CHARACTER 10
    ITEM loc CHARACTER 10
    ITEM (TABLE Emps
      ITEM eno INTEGER UNIQUE
      ITEM ename CHARACTER 10
      ITEM sal REAL)

```

In order to simplify the definition of nested schemes, we allow for the definition of *relation schemes* separately from the definition of the relations themselves. This option is analogous to the option in Pascal of defining the type of a variable directly, or by using a user-defined type. Therefore, we introduce the SCHEME command, which can be used to specify table definitions without

actually creating a table. This command is especially useful when deeply nested relations are being defined or when the same nested relation scheme is to appear in more than one place.

The formal definitions for our sample corporation database follow.

SCHEME

TABLE PARTSET

ITEM part INTEGER UNIQUE NOT NULL

TABLE PERSON

ITEM name CHARACTER 10 UNIQUE

ITEM dob CHARACTER 8

TABLE EMPLOYEE

ITEM empno INTEGER UNIQUE

ITEM name CHARACTER 10

ITEM sal REAL

ITEM mgr INTEGER REFERENCES EMPLOYEE.empno

ITEM (TABLE Children PERSON)

SCHEMA

TABLE Corp

ITEM dno INTEGER UNIQUE

ITEM dname CHARACTER 10

ITEM loc CHARACTER 10

ITEM (TABLE Emp EMPLOYEE)

ITEM (TABLE Usage PARTSET)

TABLE Supply

ITEM supplier INTEGER UNIQUE

ITEM (TABLE Supplies PARTSET)

One noticeable absence from our language is the SQL CREATE INDEX command. Indices are in the realm of physical database access concerns and should not be a user specified option. Unfortunately, in SQL, this command is also the means used to specify the UNIQUE constraint on attributes. In SQL/NF, this constraint has been moved to its rightful place in the schema definitions

and so the `CREATE INDEX` command is no longer necessary at the user level.

8.4 Comparison with Other Languages

In this section, we look at other database languages which have been developed to deal with databases that are not based on the standard 1NF model. We only briefly mention non-SQL-like languages and provide a more detailed comparison of the SQL-like languages.

Non-SQL-like languages include those developed for functional data models [Zan5, Shi] and those developed from a "Query-by-Example" model [JW, Hsi]. The GEM language [Zan5] is a derivative of QUEL which works on a semantic data model of the Entity-Relationship type. The DAPLEX language [Shi] uses an English-like syntax which works on a functional data model. Both GEM and DAPLEX use a functional composition notation to relieve users of explicitly specifying joins. This composition is explicitly represented in the n -1NF data model with the use of nested relations. GEM allows single attributes to be set-valued one level deep. For example, an attribute *color* may have value {green} or {yellow, red}. This corresponds to limiting rules in our model to the form $R = (A_1, A_2, \dots, A_n)$ where each A_i is either zero order or a higher order attribute with associated rule $A_i = (B)$ where B is zero order. Neither GEM nor DAPLEX supports explicit nesting or unnesting of set-valued attributes, however, each retains a version of the SQL `GROUP BY` operation for executing aggregate functions.

The language, Unified Query-By-Example (UQBE) [Hsi], is based on Jacobs' database logic [Jac1] and the functional data model. UQBE queries are translated into either QBE or a Functional Query Language which can be translated into other languages like QUEL and SEQUEL. Jacobs' own QBE-like language, Generalized Query-By-Example (GQBE) [JW], is based strictly on database logic. These languages operate on $\neg 1NF$ relations, however, the two-dimensional format is quite different from a SQL-like language, so direct comparison is not made. In fact, Jacobs has defined a Generalized SQL (GSQL) language [Jac2] with power similar to the QBE-like languages. We will look at GSQL later in this section.

One SQL-like language which also uses a form of functional composition is SQL/N [Bra]. SQL/N is upwardly compatible with SQL and provides "natural language" quantifiers, like "FOR ALL" and "THERE IS 1", for joining relations over common attributes. "PARENT" and "CHILD" relationships between tuples are based on the foreign key concept. As we mentioned above, the $\neg 1NF$ model allows explicit representation of these relationships with the use of nested relations.

In the rest of this section, we provide more detailed comparisons of SQL/NF with two languages designed for $\neg 1NF$ databases, GSQL and the database language being developed at IBM Heidelberg for "NF²" relations [PHH, PT, SP]. Figure 8-3 shows some example queries, written in SQL/NF, GSQL,

SQL/NF	GSQL	NF ²
1. SELECT dname, (SELECT eno, ename FROM Emps WHERE sal > 35000) FROM Company	1. SELECT (dname, E) AS DE (eno, ename) AS E FROM Company WHERE sal > 35000	1. SELECT † X.dname, (SELECT † XX.eno, XX.ename‡ FROM XX IN X.Emps WHERE XX.sal > 35000)‡ FROM X IN Company
2. SELECT dno, COUNT(Emps) FROM Company	2. No translation known	2. SELECT † X.dno, CARD(X.Emps)‡ FROM X IN Company
3. NEST Company ON ALL BUT loc AS Depts	3. SELECT (loc, Depts) (dno, dname, Emps) AS Depts FROM Company	3. SELECT † X.loc, (SELECT † Y.dno, Y.dname, Y.Emps‡ FROM Y IN Company WHERE Y.loc = X.loc)‡ FROM X IN Company
4. UNNEST Company ON Emps	4. SELECT dno, dname, loc, eno, ename, sal FROM Company	4. SELECT † X.dno, X.dname, X.loc, Y‡ FROM X IN Company, Y IN X.Emps
5. SELECT dname, ename FROM (UNNEST Company ON Emps)	5. SELECT dname, ename FROM Company	5. SELECT † Y.dname, Y.ename‡ FROM X IN Company, Y IN X.Emps

Queries:

1. Get department names and employee numbers and names for all employees making more than \$35,000.
2. Get department numbers and the number of employees in each department.
3. Create a new nested relation called *Depts* which contains all departments in each location.
4. Get the 1NF version of *Company*.
5. Get pairs of department names and employee names where the employee works in the department.

Figure 8-3. Five sample queries written in SQL/NF, GSQL, and NF² Query Language.

and the NF² query language.

8.4.1 Generalized SQL

The GSQL language is a generalization to database logic of relational SQL.

Nested relations are called *clusters*. GSQL does not support nested SFW-expressions in the FROM or WHERE clauses. All WHERE clause predicates, whether they apply to a nested relation or not, are included in the single WHERE clause of each query. If a predicate references an atomic attribute of the database relation then entire tuples are selected or rejected, however, if the attribute is in a nested relation then tuples from that nested relation are selected or rejected. In the SELECT clause, attributes may be included from anywhere in the relation. If some attributes are from nested relations, they are unnested appropriately. The attributes selected can be re-nested in an arbitrary way by specifying clusters in the SELECT clause (see query 1 in Figure 8-3.) Functions may be specified as in standard SQL, however, no support for GROUP BY is mentioned in [Jac2].

The major disadvantage of GSQL is its extreme lack of orthogonality, as witnessed by the lack of nested SFW-expressions in the SELECT and FROM clauses, and the hidden unnesting that goes on when attributes are selected. Of course, the same problems with functions in SQL, are present in GSQL, since there is no change from SQL in this area.

8.4.2 Query Language for NF² Relations

The NF² query language has syntax and properties that are similar to SQL/NF. In [PT], some of the query facilities are described and were used to generate the example queries in Figure 8-3. The language includes many more built in

functions than standard SQL, including several functions to work with a "list" data structure. They also retain the "GROUP BY" function and also include an inverse operation "DUNION." There is a new syntax for "GROUP BY" which aligns it with the syntax of our NEST and UNNEST functions:

```
GROUP reference-name IN relation-name  
BY      reference-name.attribute-name
```

There is, however, no indication of how this new grouped relation is used in a query, particularly in applying aggregate operators to the groups.

Although present in an earlier draft of the language [PHH], the latest report on the NF² query language in [PT] does not include nest and unnest functions. Nesting can be simulated using a nested expression in the select clause (see NF² query 3), however, it is very cumbersome to specify. Unnesting can also be done using nested expressions, but [PT] recommends a "hierarchical join" operation as used in NF² queries 4 and 5. According to our principle of orthogonality, a distinct unnest operator should be used rather than loading the join operation with a new function.

Chapter 9

A New Approach to Nested Normal Form

In Chapter 3, we briefly described a normal form for \neg 1NF relations, called *nested normal form (NNF)*, which was first introduced in [OY1]. [OY1] gives an algorithm to obtain an NNF decomposition of a set of attributes U with respect to a set of MVDs M . The decomposition explicitly represents a set of full and embedded MVDs implied by M , and is a faithful and nonredundant representation of U . NNF relations are better than relations with the PNF property, since NNF implies PNF and eliminates also partial and transitive dependencies which appear in the relations.

The algorithms given in [OY1] to produce NNF, use as input dependencies a set of MVDs and the MVD counterparts of a set of FDs. The authors acknowledge the deficiency which this approach to FDs creates in the design and provide a framework for a unified approach to MVDs and FDs in [YO]. The key idea is to modify the set of MVDs which are used as input to the decomposition algorithm so the different semantics of the FDs are appropriately accounted for.

Example 9.1: Let $U = ELSC$ and $D = \{E \twoheadrightarrow S, E \rightarrow L\}$, where E is an employee id, L is the employee's location, S is an employee's skill, and C is an employee's child. Using the method of [OY1], we would use the MVD $E \twoheadrightarrow L$

implied by $E \rightarrow L$ and create the \neg 1NF relation with scheme tree shown in Figure 9-1a.

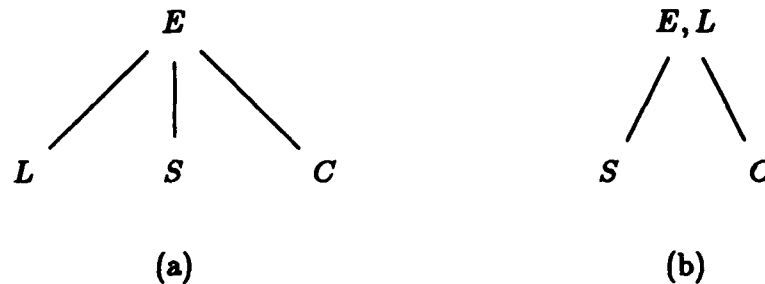


Figure 9-1. Scheme trees for Example 9.1 using approach (a) of [OY1], and (b) modified for different FD semantics.

However, since $E \rightarrow L$, each L -set created by this scheme will be a singleton set. Therefore, we should use the scheme tree of Figure 9-1b. \square

Although an approach to better handling FDs in an NNF design is being pursued [Ozs], the introduction of embedded MVDs has not yet been considered. One reason for this is that the implication problem for EMVDs has not been solved. That is, given a finite set of attributes U , there is no known complete axiomatization of EMVDs. Furthermore, as we mentioned in Chapter 2, if the set of attributes is infinite, then there is provably no complete axiomatization. There are, however, several sound inference rules for EMVDs and for EMVDs together with MVDs and FDs. Thus, if we are given that an EMVD should hold in our database, we can use that knowledge, plus any dependencies derivable from the known inference rules, to improve our database

design. One of the contributions of our new approach to NNF design is to include EMVDs in the set of input dependencies.

In addition to including EMVDs in the design, we take a different approach to the design of NNF relations, which gives the designer of a \rightarrow 1NF scheme more control over the final outcome. As proved in [OY1], the design scheme produces a unique result if and only if the input dependencies are conflict free[†]. Furthermore, we are guaranteed that the path set of the scheme trees created is in 4NF only if the input dependencies are conflict free. Otherwise, there are several designs which will satisfy the NNF requirements, not all of which will have 4NF path sets.

In the approach of [OY1], these different designs result by using different selections of fundamental keys to decompose a set of attributes into several branches of the scheme tree, and by using different orderings of all keys to test for partial and transitive dependencies and essential dependents. These different orderings of keys may cause different scheme trees to be split apart. The following example will make this explanation clearer.

Example 9.2: Consider a scouting database with attributes BSL (boy scout leader), GSL (girl scout leader), Boy, Girl, Date (when a Boy and Girl went out to eat), and Dance (when a Boy and Girl went to a dance). The dependencies which are assumed to hold in this database are $BSL \twoheadrightarrow Boy$, GSL

[†] Please refer to sections 4 and 5 of Chapter 2 for the explanation and definition of many terms used in this chapter concerning dependencies and normal forms.

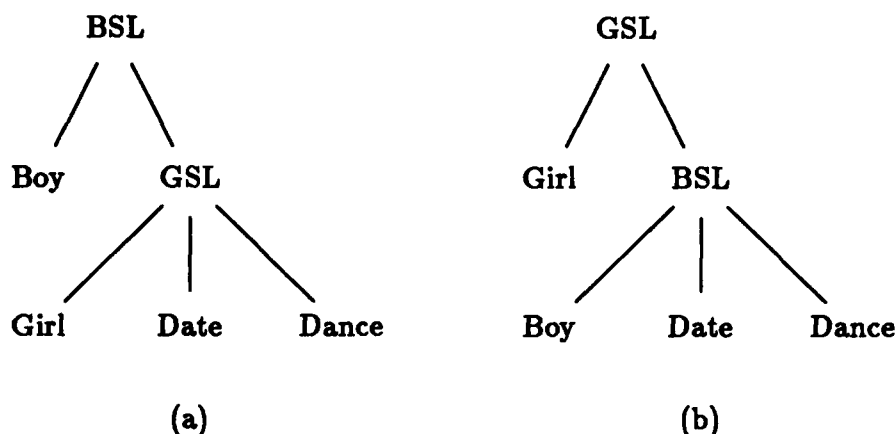


Figure 9-2. Two initial scheme trees for Example 9.2, using (a) BSL, and (b) GSL to decompose.

\rightarrow Girl, (Boy, Girl) \rightarrow Date, and (Boy, Girl) \rightarrow Dance. We also have the EMVD $\emptyset \rightarrow$ BSL | GSL, however EMVDs are not considered in the approach of [OY1]. Following the approach of [OY1] we create an initial scheme tree by decomposing the entire set of attributes based on the fundamental keys { BSL, GSL, (Boy, Girl)}. The decomposition algorithm arbitrarily chooses either of these keys to perform the initial decomposition, and depending on which one is chosen quite different NNF designs result. The two initial decompositions which result from using BSL or GSL are shown in Figure 9-2.

When BSL is used to decompose the initial scheme tree, the tree has a partial dependency $\text{GSL} \twoheadrightarrow \text{Girl}$, and so the edge (GSL, Girl) is removed from Figure 9-2a, and a new tree created with the single edge (GSL, Girl). Similarly, when GSL is used to decompose the initial scheme tree, the tree has partial

dependency $BSL \twoheadrightarrow Boy$, and so the edge (BSL, Boy) is removed from Figure 9-2b, and a new tree created with the single edge (BSL, Boy) . The scheme trees which result in these two cases have 4NF path sets (BSL, Boy) , $(BSL, GSL, Date)$, $(BSL, GSL, Dance)$, and $(GSL, Girl)$. However, in neither case is the particular decomposition very intuitive. Take the trees which result from starting with BSL . The scheme trees show that for each boy scout leader there is a set of boys and a set of girl scout leaders. And for each girl scout leader associated with a boy scout leader there is a set of Dates and a set of Dances. Also for each girl scout leader there is a set of girls. The relationship between BSL , GSL and $Date$ and $Dance$ is only an indirect one via the leaders associated boys and girls. Nevertheless, these two schemes are the ones recommended by [OY1].

The other alternative is to use $(Boy, Girl)$ as the fundamental key to start the decomposition. However, this choice is not allowed by [OY1] since the MVDs with left hand sides $(Boy, Girl)$ are split by the other given MVDs, and this will result in a path set which is not 4NF. However, let us explore what happens if we do use this fundamental key, primarily to compare later with results achieved by our design approach for this problem. Using $(Boy, Girl)$ as the fundamental key two alternatives for the initial decomposition are shown in Figure 9-3. The two alternatives result from a decision to use either BSL or GSL as the fundamental key when decomposing node (BSL, GSL) .

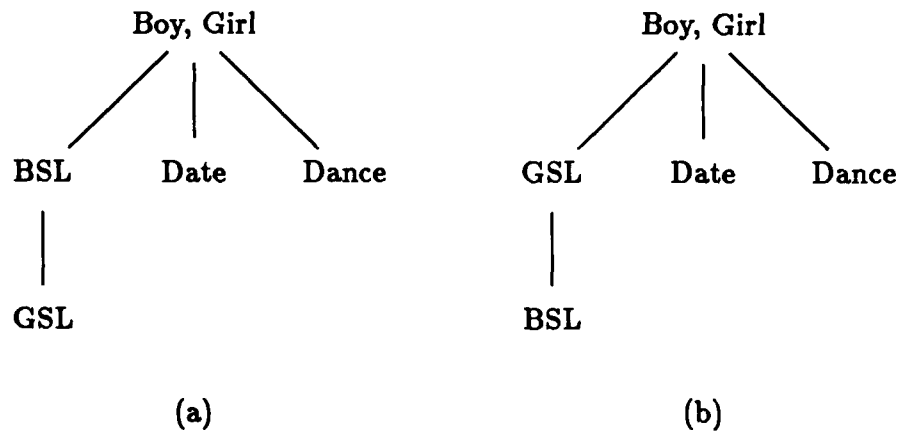


Figure 9-3. Two alternative trees using (Boy, Girl) to start decomposition, and using (a) BSL, and (b) GSL to further decompose.

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

dependency presents quite an unintuitive grouping of attributes. □

In our approach, the design algorithm will start with a 4NF decomposition and will preserve that decomposition throughout the remainder of the design. Thus, the primary point where different NNF designs will originate is embodied in the well studied and understood creation of a 4NF decomposition. We note, that when the input set of dependencies is conflict free there is a unique 4NF decomposition, and, therefore, our approach also produces a single NNF design for this case. Let us consider Example 9.2 using a preview of our approach.

Example 9.3: (Continuation of Example 9.2.) We saw that it seemed best to ensure that Date and Dance were associated with the key (Boy, Girl) and so in the 4NF decomposition we use the key (Boy, Girl) to make the first split. Thus, we decompose into schemes (Boy, Girl, Date), (Boy, Girl, Dance), and (Boy, Girl, BSL, GSL). Now we can use the other two MVDs in any order to decompose (Boy, Girl, BSL, GSL) into (BSL, Boy), (GSL, Girl), and (BSL, GSL). Considering just the MVDs, this decomposition is in 4NF. If we consider the EMVD, as we propose to do, then the scheme (BSL, GSL) is decomposed into BSL and GSL, and these two schemes are eliminated since they are proper subsets of other schemes. Our method will then proceed to create a scheme tree for each scheme in the 4NF decomposition, as shown in Figure 9-4.

Then we combine scheme trees when the common attributes of two

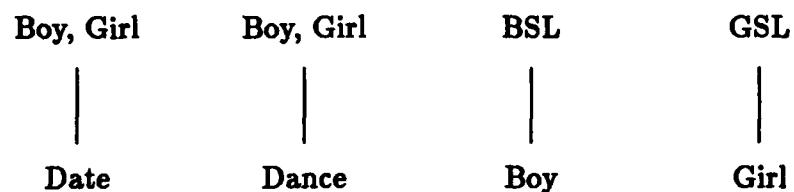


Figure 9-4. Initial scheme trees for each 4NF scheme of Example 9.3.

trees form the same root to non-leaf path in both trees. In this example, we combine the two trees with root (Boy, Girl) and our final design is a set of three scheme trees as shown in Figure 9-5. This design more clearly depicts the intended relationships and came about partially due to the fact that we carefully selected the 4NF decomposition that was appropriate for this case. In the approach of [OY1], this kind of decision making can only go into the choice of fundamental key selection, and there is no way to produce the scheme trees of Figure 9-5, no matter what choices are made. We note that if we did not allow the EMVD to influence our 4NF decomposition, then we would have had an additional edge relating BSL and GSL in either the BSL-Boy tree or the GSL-Girl tree. These trees would still be more intuitive, and are equally unattainable using [OY1]. \square

9.1 Definitions and Basic Procedures

The first procedure we will need for our algorithm is a 4NF decomposition procedure. Several have been proposed, however we require one that deals with both FDs and MVDs and does not treat FDs as MVDs in the design,

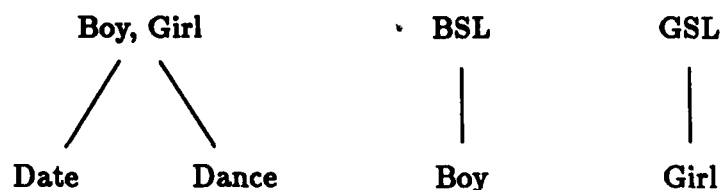


Figure 9-5. Final scheme trees using new approach to design of Example 9.3.

thereby ignoring the different semantics that FDs impose. Two approaches are available, one by Beeri and Kifer [BeK1] and Katsuno [Kat], and the other by Yuan and Özsoyoğlu [YO]. In the first approach, given a set D of FDs and MVDs, a new set M' of MVDs is formed by first obtaining the full version of the MVDs in D , and then replacing the left-hand side X of each MVD in the full version by the closure of X with respect to D .

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

Definition 9.1: The *envelope set* $E(D)$ of a set D of FDs and MVDs is

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

If a database scheme is 4NF with respect to $E(D)$ then it is also 4NF (BCNF if D has FDs only) with respect to D . Thus, a database scheme for D can be obtained by using $E(D)$ as input to any 4NF decomposition algorithm [Fag2,

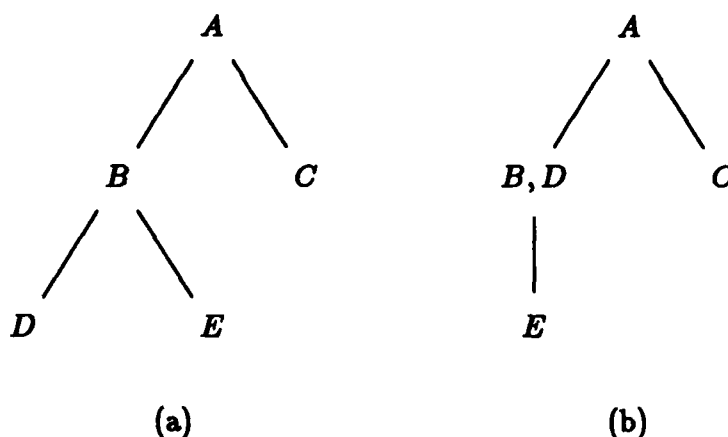


Figure 9-6. Scheme tree (a) T_1 , and (b) T_2 .

Lie1, GR, OY1], without considering the FDs and MVDs separately.

We must consider which of these two approaches to the design of flat databases will help us most in forming better $-1NF$ designs. As shown in Example 8.1, in an NNF design FDs cause singleton sets to appear if the MVD represented by an edge in a scheme tree is also an FD. In general, nesting is not necessary when for some edge (u, v) , $u \rightarrow v$ holds. Consider the scheme tree T_1 shown in Figure 9-6a. If $B \rightarrow D$ holds, then each B value will have a single D value associated with it. Therefore, there is no need to nest D values allowing the tree T_2 , shown in Figure 9-6b, which has a smaller structure and is consistent with T_1 in that $MVD(T_1) \Rightarrow MVD(T_2)$.

In the first approach described above, a similar operation takes place in the closing of the left hand sides of the MVDs. Attributes in the depen-

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

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

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

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

Since we desire to include EMVDs in our design algorithm, we must perform some additional steps to achieve our final decomposition. Let U be the set of attributes to be used in the design, D a set of given FDs and MVDs, and

F a set of given EMVDs. First, using the known inference rules, we generate all FDs, MVDs, and EMVDs that are implied by the EMVDs or the EMVDs together with any known FDs or MVDs. The new FDs and MVDs are added to the original set D and the new EMVDs are added to the original set F . We compute M' , the set of MVDs obtained by making the MVDs implied by D full and closing the left hand sides. We also close the left hand sides of the EMVDs in F using the FDs in D . We then use M' as input to one of the usual 4NF decomposition algorithms. This results in a set of schemes $\mathcal{R} = \{R_1, R_2, \dots, R_n\}$.

Each scheme in \mathcal{R} may have one or more FDs implied by D embedded within it. In their flat database design, [BeK1] use these FDs to synthesize a set of schemes for each scheme of \mathcal{R} , further eliminating redundancy by achieving a 3NF decomposition. We do not want to take the additional step of synthesizing 3NF schemes when designing \neg 1NF relations since organizing these schemes in a scheme tree will only introduce singleton sets. However, when we allow EMVDs to influence our design, we will have to consider further decomposition based on the FDs in each scheme of \mathcal{R} . The reason for this is that we can not use an EMVD in the design process unless at some stage in the decomposition it becomes a full MVD. For example, if $U = ABCDE$, then we can not use the EMVD $A \twoheadrightarrow B|CD$ until U is decomposed into a scheme which does not have E in it. Thus, we perform two checks for the EMVDs in F following our decomposition into \mathcal{R} with respect to M' . First, if an EMVD, F_j , becomes a

nontrivial MVD when F_j is projected onto some scheme in \mathcal{R} , say R_i , then F_j is used to decompose R_i , and we replace R_i with its decomposition in \mathcal{R} . This process is repeated until no more EMVDs can be used to decompose schemes in \mathcal{R} . Second, for each scheme R_i in \mathcal{R} we perform a temporary 3NF synthesis on R_i obtaining the schemes $S = S_1, S_2, \dots, S_k$. If an EMVD, F_j , becomes a nontrivial MVD when F_j is projected onto some scheme in S , say S_j , then we use F_j to decompose S_j and add the decomposition to \mathcal{R} . This continues until all schemes in S have been considered. If any schemes remain in S then we take the union of those schemes and replace R_i with this union. If all schemes still remained in S then we replaced R_i with R_i and no change was made to \mathcal{R} . The remainder of the \neg 1NF design uses M' , F , and \mathcal{R} .

Example 9.4: [Ull] Let $U = SPYC$, $D = \{SP \rightarrow Y\}$, and $F = \{C \twoheadrightarrow S|P\}$, where C is a course taken by a student S , and the course has prerequisite P taken by the student in year Y . There are no nontrivial FDs or MVDs implied by the EMVD, so we find $M' = \{SPY \twoheadrightarrow C\}$. Using this set as input to a 4NF decomposition algorithm, we get the scheme $SPYC$. Since this is the original set U , the EMVD is still an EMVD for this scheme. In the next step, we perform a synthesis on this scheme and get the decomposition $\{SPY, SPC\}$. Since, the EMVD in F is an MVD for scheme SPC , we use it to decompose SPC into CS and CP . The final decomposition is $\{CS, CP, SPY\}$, and using our NNF algorithm we get the scheme trees shown in Figure 9-7a. In comparison, the scheme tree produced by the NNF algorithm of [OY1] is shown in Figure 9-7b.

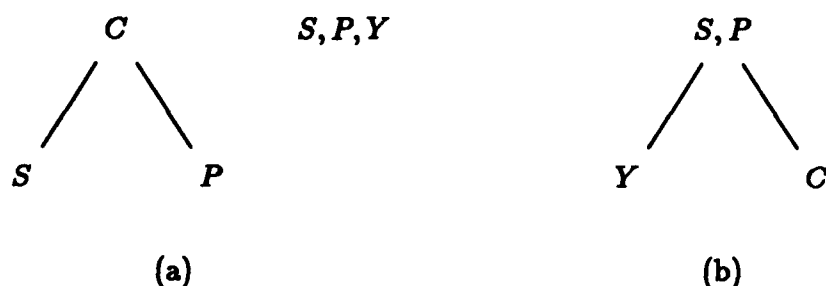


Figure 9-7. Scheme trees for Example 9.4 using (a) our approach, and (b) [OY1].

□

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

In order to maintain NNF, even in a trivial design, we can not decompose each 4NF scheme arbitrarily. This is due to the requirement that only fundamental keys (see section 3.4.2) be used as the non-leaf nodes of a scheme tree. Thus, we use a simplified version of the DECOMP procedure in [OY1] to perform the decomposition. The procedure is much simpler since the input is a set of attributes forming a 4NF scheme and there is no possibility of a split key appearing among these attributes (a condition checked for in the original

procedure), and there is exactly one dependent in the dependency basis of any attribute set which is a subset of a 4NF scheme. We first provide a new definition for "fundamental keys," since we also need to deal with the set F of EMVDs which hold in the database. We also improve the definition by preferring fundamental keys which are not projections of some essential key. For example, if A and BC are essential keys and if we are finding the fundamental keys of ABD , then we would prefer to decompose based on the fundamental key A rather than B , since A represents a more complete relationship than B which is a projection of BC .

Definition 9.2: Given a set M' of MVDs, a set F of EMVDs, and a set U of attributes with $V \subseteq U$, the set of *candidate fundamental keys* of V , denoted $CFK(V)$, is defined as follows:

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

Out of $CFK(V)$ we prefer those keys that are minimal subsets of V and if there are none, we use the minimal intersections of those keys with V . The *preferred fundamental keys* of V , denoted $PFK(V)$, and all *fundamental keys*

of V , denoted $FK(V)$, are defined as follows:

$$PFK(V) = \{X | X \in CFK(V) \wedge X \subseteq V \wedge$$

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

$$FK(V) = \{W | X \in CFK(V) \wedge W = X \cap V \wedge W \neq \emptyset \wedge$$

$$\nexists Y \text{ such that } Y \in CFK(V) \wedge Y \cap V \subset W\}.$$

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

Procedure DECOMP(V, T)

{ V is a set of attributes which is a node in scheme tree T }

begin

If V has 2 or more elements and $FK(V) \neq \emptyset$ then

begin

(1) If $PFK(V) \neq \emptyset$ then let $V_0 \in PFK(V)$

else let $V_0 \in FK(V)$;

(2) $W = V - V_0$;

(3) Change V into V_0 in T ;

(4) Attach W as a son of V_0 ;

(5) DECOMP(W, T);

end

end.

The final procedure we need for our design algorithm is a method for combining scheme trees while maintaining NNF and the original 4NF decomposition. We can combine scheme trees if the attributes that are in common to both trees form the same path, u to v , in each tree, where u is the root and v is a non-leaf node in both trees. If we have two trees that meet this requirement then we can temporarily merge them into a single tree. If the merged tree

is free of transitive dependencies, then we let the merge become permanent. After making all possible merges, we have our final NNF design. The MERGE procedure is as follows:

```

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

```

9.2 The NNF Design Algorithm

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

Algorithm 2

```

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

```

(e) For $i := 1$ to k do
 begin
 (i) Synthesize a 3NF decomposition of R_i , $S = (S_1, S_2, \dots, S_m)$.
 (ii) Decompose schemes in S according to any EMVDs in F which project as nontrivial MVDs on some scheme in S . Remove any decomposed scheme from S and add the decomposition to \mathcal{R} .
 (iii) Replace R_i in \mathcal{R} with the union of the remaining schemes in S .
 end
 (2) Prepare initial scheme trees.
 (a) Initialize k scheme trees T_1, T_2, \dots, T_k with no edges and single nodes labeled R_1, R_2, \dots, R_k , respectively.
 (b) For $i := 1$ to k do DECOMP(R_i, T_i) end.
 (c) Let $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$.
 (3) Merge trees.
 Until no more changes can be made to \mathcal{T} do
 begin
 (a) Select $T^1 \in \mathcal{T}$ and $T^2 \in \mathcal{T}$, $T^1 \neq T^2$, such that T^1 and T^2 have not been considered together.
 (b) If the common attributes of T^1 and T^2 from the same root to non-leaf path in both trees then
 begin
 (i) MERGE(T^1, T^2, T^3).
 (ii) If there are no transitive dependencies in T^3 then
 $\mathcal{T} := \mathcal{T} - \{T^1, T^2\} \cup \{T^3\}$
 end
 end
 end.
 end.

9.3 Correctness of Algorithm 2

In this section we show that the 3NF design produced by Algorithm 2 is in nested normal form. To do this we need to show that the four requirements of NNF hold for each relation in the design. Each scheme tree T of the design

must satisfy the following four properties:

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

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

We will prove these four properties hold after each major step of Algorithm 2 in which scheme trees are created or modified.

Proposition 9.1: The four properties of NNF hold after step 2 of Algorithm 2 where the initial scheme trees are created.

Proof:

- (1) Inference property: Since all MVDs and EMVDs in $MVD(T)$ are trivial for the single path trees which procedure DECOMP produces, this property holds trivially.
- (2) PD property: Assume there is a partial dependency in T . Then the path set of T can be decomposed using the partial dependency, and

therefore is not in 4NF. This contradicts the fact that we start with all path sets being in 4NF as a result of step 1 of Algorithm D.

- (3) TD property: Trivially true, since the definition of transitive dependency requires sibling nodes to exist in the tree, and there are none in a single path tree.
- (4) FK property: Procedure DECOMP creates non-leaf nodes which are fundamental keys of the subtrees with those nodes as root. Thus, this property holds by design. \square

Proposition 9.2: The four properties of NNF hold after step 3 of Algorithm 2 where scheme trees are merged.

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

- (1) Inference property: Figure 9-8 shows two general trees T_1 and T_2 with common attributes u_1, u_2, \dots, u_n forming the same root to non-leaf path in both trees. Subtrees are summarized by the union of all nodes in the subtree (e.g., Y_1^2, Z_1^1). Each of these trees is assumed to be in NNF. Given that these trees are in NNF and the path sets are in 4NF, the following JD holds:

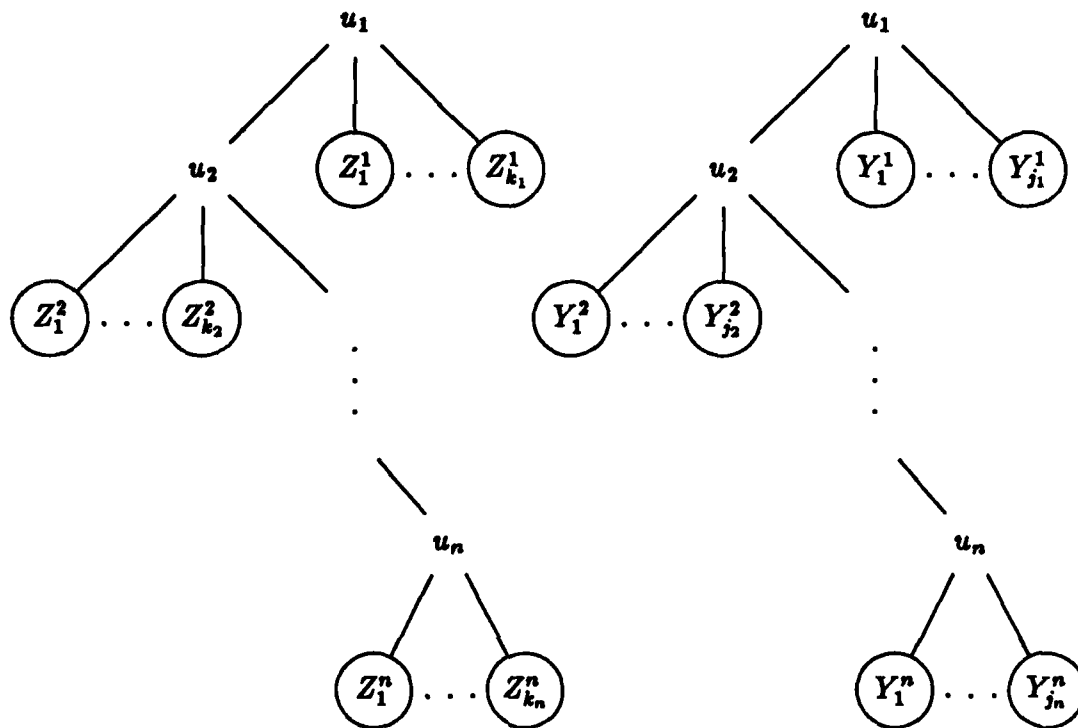


Figure 9-8. General trees T_1 and T_2 .

$$\begin{aligned} \times & (u_1 Z_1^1, \dots, u_1 Z_{k_1}^1, u_1 u_2 Z_1^2, \dots, u_1 u_2 Z_{k_2}^2, \dots, u_1 u_2 \dots u_n Z_1^n, \dots, u_1 u_2 \dots u_n Z_{k_n}^n, \\ & u_1 Y_1^1, \dots, u_1 Y_{j_1}^1, u_1 u_2 Y_1^2, \dots, u_1 u_2 Y_{j_2}^2, \dots, u_1 u_2 \dots u_n Y_1^n, \dots, u_1 u_2 \dots u_n Y_{j_n}^n, \\ & S(T_3), \dots, S(T_m)). \end{aligned}$$

This JD implies

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

holds in T' which is the EMVD representing edge (u_1, u_2) . Similarly, the JD implies each edge (u_i, u_{ℓ}) , $1 \leq i = \ell - 1 \leq n - 1$. Also,

the EMVDs represented by each edge in the Y and Z subtrees are still implied by this JD. Therefore, $MVD(T')$ holds and the inference property is maintained.

- (2) PD property: Holds as in Proposition 8.1, since we have not modified the 4NF path sets by merging T_1 and T_2 .
- (3) TD property: By design, we specifically test that this property is not violated before we merge trees permanently.
- (4) FK property: Even though some of the non-leaf u_i nodes may have a new set of descendants consisting of the Y^i and Z^i subtrees at that level, u_i will still be a fundamental key of $V = u_i Y_1^i Y_2^i \dots Y_{j_i}^i Z_1^i Z_2^i \dots Z_{k_i}^i$. If u_i was a minimal intersection of a subset of V and the keys of M' , then it will be minimal for V . \square

By Propositions 8.1 and 8.2, we know that relations designed using Algorithm 2 will be in nested normal form with respect to M' and F . Since $D \Rightarrow M'$, and $M' \cup F \Rightarrow MVD(\mathcal{T})$, we have a good representation of the original dependencies in our design.

9.4 Further Normalization of NNF Relations

Algorithm 2 produces a set of 4NF relations which is in NNF with respect to a set of MVDs (M') and a set of EMVDs (F). M' was derived from a set D of MVDs and FDs in step one of the algorithm. Later steps deal only with M'

and F , and ignore the FDs that existed in D . This was appropriate since we incorporated the FDs by closing the left hand sides of the MVDs in D to obtain M' . This eliminates the possibility of getting nested relations which will only have a single tuple in them. However, there is still a place where redundancy due to FDs arises in our $\neg 1NF$ design. The following example will illustrate this problem.

Example 9.5: Consider the following university database taken from [Lie1]. We have attributes Class, Day, Hour, Tutor, Office, Student, Major, and Exam. The dependencies which hold are

$$\text{Class} \twoheadrightarrow \text{Day}$$

$$\text{Student} \rightarrow \text{Major}$$

$$\text{Tutor} \rightarrow \text{Office}$$

$$\text{Class, Student} \twoheadrightarrow \text{Exam}$$

$$\text{Class, Tutor} \twoheadrightarrow \text{Hour}$$

Following Algorithm 2, we make the MVDs full and close their left hand sides giving M' :

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

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

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

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

$$\text{Class, Day}$$

$$\text{Class, Student, Major, Exam}$$

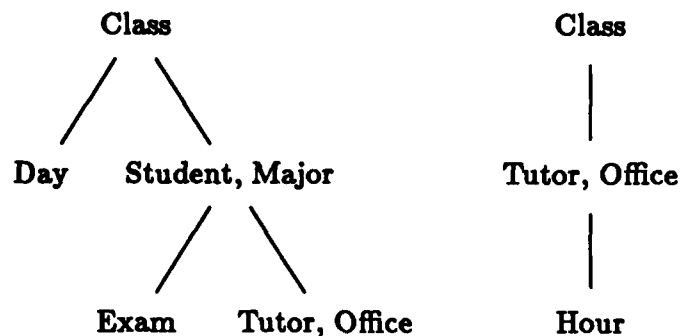


Figure 9-9. Scheme trees produced by Algorithm 2 for Example 9.5.

Class, Tutor, Office, Hour

Class, Tutor, Office, Student, Major

One of the two symmetric choices that Algorithm 2 produces for this set of schemes is shown in Figure 9-9.

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

The problem with designs, such as those in this example, is that groups of attributes which are nested will introduce redundancies if a subset of that

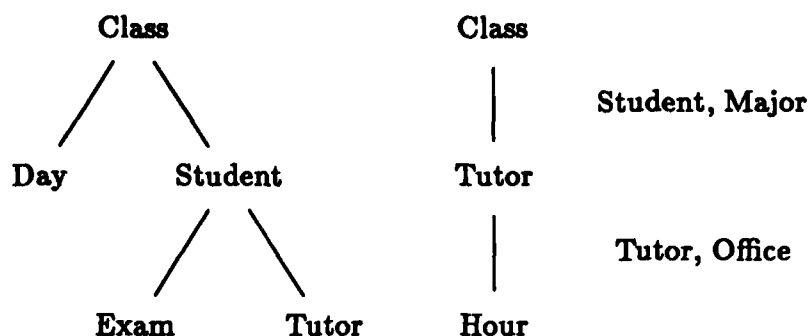


Figure 9-10. Scheme trees produced by Algorithm 3 for Example 9.5.

group functionally determines some other part of the group. Note that this problem does not occur if the group is at the root of the scheme tree, since these are the atomic attributes of the relation and their values will occur only once in the relation.

The solution is to examine each scheme tree produced by Algorithm 2 for nodes N that exhibit the above behavior, replacing each N with the smallest set of attributes which functionally determines N , and creating a new relation with a single node containing the attributes involved in the redundant FD as root and no branches. For Example 9.5, the new design would consist of the four scheme trees shown in Figure 9-10. Below we give a new algorithm to implement these changes.

Algorithm 3

{ input: a set \mathcal{T} of scheme trees produced by Algorithm 2
 and the set of FDs G used as input to Algorithm 2
 output: a new set \mathcal{T} of scheme trees in NNF
 with redundancies due to FDs removed. }

```

begin
  Until no more changes can be made to  $\mathcal{T}$  do
    begin
      If there exists  $T \in \mathcal{T}$ , where  $T$  contains a node  $N$  such that
         $X \rightarrow N$  is implied by  $G$ , with  $X \subset N$  then
          begin
            (a) Let  $Z \subset N$  where  $G \Rightarrow Z \rightarrow N$ , and
                for no  $W \subset Z$  does  $G \Rightarrow W \rightarrow N$ .
            (b) Let  $Y \subset Z$  where  $G \Rightarrow Y \rightarrow N - (Z - Y)$ , and
                for no  $W \subset Y$  does  $G \Rightarrow W \rightarrow N - (Z - Y)$ .
            (c) Modify  $T$  by replacing node  $N$  with  $Z$ .
            (d) Add a new tree to  $\mathcal{T}$  with the single node  $N - (Z - Y)$ 
                and no edges.
          end
        end
      end
    end.

```

It is straightforward to show that the scheme trees produced by Algorithm 3 are still in NNF with respect to M' , and if we consider the FDs used to change the trees then the path sets are still in 4NF, and so the join dependency among the path sets continues to hold.

Chapter 10

Conclusion

In this chapter we summarize the results presented in this dissertation, and provide direction for future work in this area.

10.1 Summary of Results

Dropping the 1NF assumption in relational databases is not a trivial step to take. The added complexity requires a thorough reexamination of the body of relational database theory that has already been developed for 1NF databases. In addition, there is ample opportunity for exploring new techniques specific to a \neg 1NF model. One advantage of our \neg 1NF model is the orthogonality of the change made to the 1NF model. Instead of allowing just any data structure to model the decomposable values that we now permit in relations, we choose the relation as that structure. This allows a large bulk of the current theoretical results for relational databases to be applied in a recursive manner to \neg 1NF databases. As we illustrated in Chapter 3, the extension to allow nested relations is quite adequate for modeling a large variety of database problems, as well as improving the design of traditional databases. As a result of our research, we make several important contributions in the \neg 1NF relational database area.

In Chapter 4, we defined an extended relational calculus for use with

\neg 1NF relations. This calculus forms a theoretical basis for the expressive power of a \neg 1NF query language. In Chapter 6, we found that the calculus is equivalent to the basic relational algebra extended with nest and unnest operators, thus verifying the power of extended algebras proposed by other researchers [JS, FT]. In Chapter 5, we defined a normal form for \neg 1NF relations, called partitioned normal form. Just as traditional relations are assumed to be in 1NF, we believe that \neg 1NF relations should be assumed to be in PNF. PNF is a basic normal form which states that the atomic-valued attributes of a relation are a key for that relation. This assumption has several desirable consequences. First, we have the intuitive semantics that more than one nested set of values should not be associated with the same set of atomic values. If that seems to be the case, then there is one or more hidden attributes that have not been properly included in the database design. Second, PNF relations always have the property that nest is an inverse for unnest, whereas, in general, that is not true. This means that information is preserved when restructuring relations. Finally, it is straightforward to define a set of extended relational algebra operators which are closed under the set of PNF relations, and maintain the data dependencies which underlie the structure of each PNF relation. In addition, these operators are faithful and precise generalizations of the standard operators with respect to unnesting. Therefore, we can use an extended operator to achieve the same result as unnesting a relation, applying the standard operator, and then reneesting.

Since the \neg 1NF model allows us to represent multiple relationships in a single relation without the redundancy that exists in a 1NF model, it is critical that we allow null values in our model. This way, if one relationship is unknown or nonexistent, then we can still store another relationship that is known. In Chapter 7, we examined the role of null values in a 1NF model, and extended those results to the \neg 1NF model. One of the critical issues here was how to handle the empty nested relation. We found that the empty nested relation is equivalent to a relation with a single null tuple in it. A null tuple consists of no-information null values for the atomic-valued attributes and, recursively, nested relations with single null tuples for the set-valued attributes. This means that unnest is still defined properly even when unnesting an empty nested relation. Unlike other approaches which eliminate tuples when an empty nested relation is unnested, our approach preserves information in the relation. We gave also new definitions for the extended algebra operators so that they deal with null values. We found that these null-extended operators are faithful and, in the case of union and projection, precise generalizations of the standard operators with respect to unnesting and an open world possibility function. We found a null-extended natural join which is an adequate and restricted generalization of standard natural join and a null-extended difference which is an adequate and restricted generalization of standard difference. Finally, we found that arguments which have led to new axiomatizations for functional and multivalued dependencies in the presence of null values are based on incorrect assumptions

about the nature of null values. We showed that the usual axiomatizations are valid, and should be used for dependency inference even when null values are present.

In Chapter 8, we defined a \neg 1NF user language, called SQL/NF, which is based on the commercial database language SQL. SQL is a powerful query language, responsible for a lot of the current acceptance of relational databases. SQL-like languages are easy to learn and provide improved data independence over former database query languages. We have extended SQL to enhance its ease of use and expanded its expressiveness to deal with \neg 1NF relations. Since our extensions keep the relational model “pure” in that all data is represented as relations, or, recursively, as relations within relations, there are no longer two types of structures—relations and partitioned relations (created by “GROUP BY”.) This consistency makes application of functions and use of SFW-expressions straightforward and logical. In summary, the major advantages of our language are

- (1) Orthogonality of expressions. Wherever a relation could logically occur, a SFW-expression is allowed.
- (2) Orthogonality of functions. Functions are applied to relations, and not to attributes which stood for relations.
- (3) Arbitrary restructuring of relations via the nest and unnest operators.

- (4) Elimination of "GROUP BY" and "HAVING" clauses.
- (5) Use of references names to simplify queries and to rename attributes.
- (6) More complete and logical treatment of null values, including a method for performing outer joins and elimination of subsumed tuples.
- (7) Upward compatibility from a strict 1NF system, in which SFW-expressions in the SELECT clause must evaluate to single values, and relation-values are not allowed in the database.

Finally, we looked at the design of \neg 1NF relations using the criteria of nested normal form. NNF eliminates anomalies due to partial and transitive redundancies in PNF relations. In Chapter 9, we presented a new algorithm for achieving an NNF design. Our approach has the advantage of using a 4NF decomposition as input to the algorithm. This gives the user more control over the final design of the \neg 1NF relations, by allowing him to choose the 4NF decomposition which emphasizes the data associations he considers most critical. We made also several improvements to the design process by considering embedded multivalued dependencies in coming up with the 4NF decomposition, and by better utilizing functional dependencies in the design of the \neg 1NF relations. Functional dependencies are especially important in that a naive approach to their use will create many single tuple nested relations and unnecessary redundancy in the design.

10.2 Directions for Future Work

We feel there are three primary areas where future work is necessary for $\neg 1NF$ relations. These areas are extensions to the model to include recursive schemes, the recursive algebra and optimization of the algebra and SQL/NF, and implementation. We briefly describe the problems in these three areas below.

10.2.1 Recursive Schemes

In the $\neg 1NF$ model defined in Chapter 3, we restrict relation schemes to be nonrecursive. A possible direction for future work is the elimination of this restriction and the study of the ensuing consequences. There are many situations for which recursive schemes would be an appropriate model. Consider a management hierarchy. This can be represented by the recursive scheme: $Employee = (name, Employee)$. In this relation scheme each employee has a name and a set of employees who work for him. The recursion stops when, at the bottom of the hierarchy, an employee has no one working for him and so his $Employee$ set is empty. Problems involve querying such a relation, referencing the same attribute at many different levels, and redundancy of the hierarchy if an employee works for more than one person. Special operators will be needed to search the hierarchy, to merge sets at different levels, and to do transitive closures [Zlo]. In [Jac1, Jac2], recursive schemes are allowed in the more powerful database logic, but operators on these schemes are not easy to formulate

or understand by database users, and the problems mentioned are not entirely solved.

10.2.2 Recursive Algebra and Optimization

In Chapter 4, we defined an extended relational algebra which made no changes to existing operators and added nest and unnest. Although this algebra is powerful enough to operate in a \neg 1NF environment, it lacks the convenience that is provided by a recursive algebra [Jae3, ScS1]. Using the extended algebra a relation must be unnested in order to perform operations on the tuples or attributes of the nested relations. In a recursive algebra, operations on nested relations may be nested within the projection operator. Although, the recursive algebra has the same expressive power as the nonrecursive algebra [Jae2, Jae3], the convenience of the recursive algebra outweighs the simple semantics of the nonrecursive algebra. In addition, the recursive algebra more closely matches the structure of SQL/NF, and so it is a more appropriate vehicle for optimizing SQL/NF queries. Research is needed to investigate optimizing recursive algebra queries and translating SQL/NF queries into the recursive algebra so they can be optimized and executed. Some optimization work involving operators similar to nest and unnest for statistical databases was done in [OMO].

10.2.3 Implementation

Some implementation work for \neg 1NF relations has been done in the Federal

Republic of Germany [BR, DGW, D+, GP, Sch2], France [AB2, B+], and the United States [Bat]. This field is still wide open for the development of new techniques for storing and accessing data in \neg 1NF relations. Storage structures, access techniques, indexing of nested relations, concurrency control, and human interfaces are all areas where more work is required to make \neg 1NF a viable alternative to existing models. Furthermore, the view mechanism for 1NF databases must be expanded for \neg 1NF databases, and the update problem for views must be reexamined. We believe that lower redundancy of data and the reduced use of join operations will more than make up for the added complexity in storing and accessing \neg 1NF relations.

Appendix A

In this appendix we prove that a powerset operation is not achievable using the operators of the extended relational algebra presented in Chapter 4. A powerset operation finds all subsets of a given set. If a set has n elements then the powerset has 2^n elements. We assume that the given set is a relation r with each tuple of r containing one of the elements of the set. Thus, r has n tuples, or $|r| = n$. Our goal for a powerset operator would be to create a relation with 2^n tuples, each tuple containing a set of values from the original relation. Our strategy is to show that there is no extended algebra expression E which can create an exponential number of tuples where E operates only on r and constant relations. We do this in two steps. First, we show that if E has k operators then the number of tuples in the relation formed by $\mu^*(E)$ is $\Theta(n^{k+1})$. Then, since E must have equal to or fewer tuples than $\mu^*(E)$, $|E|$ will also be $\Theta(n^{k+1})$. Since we must fix k in any expression, we can always find a relation r such that $2^{|r|} > |E|$, and so E can not be a powerset operation.

Lemma A-1. Given a relation r and an extended algebra expression E , $|r| = n$, E has k operators; the number of tuples in the relation formed by $\mu^*(E)$ is $\Theta(n^{k+1})$.

Proof: The proof is by induction on the number of operators k in expression E .

Base case: $k = 0$: trivial, $|\mu^*(E)| = \Theta(n)$ as E can only be r or a constant relation.

Induction step: There are two cases, one for unary operators and one for binary operators.

Case 1: Unary operator (σ, π, ν, μ) . Assume we have an expression E' with j operators where $|\mu^*(E')| = \Theta(n^{j+1})$. Consider the expression $E = \theta(E')$, where θ is one of the unary operators. The operators σ , π , and ν can not increase the number of tuples in E' and so $\mu^*(E)$ will have no more tuples than $\mu^*(E')$. Therefore, $|\mu^*(E)|$ is also order n^{j+1} and so is certainly order $n^{(j+1)+1}$. If θ is μ , then, even though $|E| \geq |E'|$, we will have $|\mu^*(E)| = |\mu^*(E')|$. This is because the explicit unnest operation performed in E is also present in the μ^* operation on E' . Thus, as for the other unary operators, $|\mu^*(E)| = \Theta(n^{(j+1)+1})$.

Case 2: Binary operator $(\times, \cup, -)$. Assume we have two expressions E' with ℓ operators and E'' with m operators, where $\ell + m = j$, $|\mu^*(E')| = \Theta(n^{\ell+1})$, and $|\mu^*(E'')| = \Theta(n^{m+1})$. If $E = E' - E''$ then there are no more tuples in E than there are in E' and so $|\mu^*(E)| = \Theta(n^{\ell+1})$, which implies that $|\mu^*(E)| = \Theta(n^{(j+1)+1})$. Cartesian product and union can increase the size of the new relation. For cartesian product the new size will be the product of the sizes of the two operands, and for union the new size will be at most the sum of the sizes of the two operands.

First, let $E = E' \times E''$. Since, $\mu^*(E' \times E'') = \mu^*(E') \times \mu^*(E'')$ [FT], we have

$$\begin{aligned} |\mu^*(E' \times E'')| &= \Theta(n^{\ell+1})\Theta(n^{m+1}) \\ &= \Theta(n^{\ell+1+m+1}) \\ &= \Theta(n^{(j+1)+1}) \end{aligned}$$

For union the result is similar, except that we sum the cardinalities of the operands, which is certainly of order of the product.

In each case we find that given an expression(s) with j operators with cardinality of order n^{j+1} , if we add one more operator then the cardinality is of order $n^{(j+1)+1}$. This proves the induction and the lemma is proved. \square

Theorem A-1. *Given a relation r , there is no expression in the extended relational algebra which can compute a powerset operation on r .*

Proof: Suppose there was an expression P that could perform a powerset. Then $|P| = 2^n$, where $|r| = n$. By Lemma A-1, we know that P must have cardinality which is of order that is polynomial in n , specifically $\Theta(n^{k+1})$, where k is the number of operators in P . Since, we can choose n such that $2^n > \Theta(n^{k+1})$, it is not possible that P computes the powerset of r . \square

Appendix B

SQL/NF BNF

The following is a modified BNF definition of the queries facilities, DML, and DDL in SQL/NF. We used RDL [X3H2] as a baseline definition. Non-distinguished symbols are enclosed with “ $\langle \rangle$ ”. The structure [] indicates an optional entry, and the structure “...” indicates an additional zero or more repetitions of the previous entry. Braces are used for grouping in the BNF. Except where modified by braces, sequencing has precedence over disjunction (indicated by “|”).

Query Facilities

\langle query expression \rangle ::= \langle query spec \rangle | \langle structured query \rangle
| function (\langle query expression \rangle)
| \langle nested query expression \rangle
| \langle query expression \rangle \langle set operator \rangle \langle query expression \rangle

\langle structured query \rangle ::= NEST \langle nested query expression \rangle ON \langle column list \rangle
[AS \langle column name \rangle]
| UNNEST \langle nested query expression \rangle ON \langle column list \rangle
| ORDER \langle nested query expression \rangle BY \langle sort spec \rangle ...

\langle sort spec \rangle ::= { \langle unsigned integer \rangle | \langle column name \rangle } [ASC | DESC]

\langle query spec \rangle ::= \langle select from spec \rangle
[WHERE \langle search condition \rangle [PRESERVE \langle table list \rangle]]

\langle select from spec \rangle ::= SELECT \langle select list \rangle FROM \langle table list \rangle | \langle table name \rangle

\langle select list \rangle ::= ALL | \langle column list \rangle | \langle select spec \rangle [{, \langle select spec \rangle }...]

\langle select spec \rangle ::= \langle column expression \rangle | \langle reference name \rangle .ALL

\langle column expression \rangle ::= \langle value expression \rangle [AS \langle column name \rangle]

\langle table list \rangle ::= \langle table spec \rangle ...

\langle table spec \rangle ::= \langle nested query expression \rangle [AS \langle column name \rangle]

\langle search condition \rangle ::= \langle boolean term \rangle | \langle search condition \rangle OR \langle boolean term \rangle

<boolean term> ::= <boolean factor> | <boolean term> AND <boolean factor>
 <boolean factor> ::= [NOT] <boolean primary>
 <boolean primary> ::= <predicate> | ((search condition))
 <predicate> ::= <comparison predicate> | <between predicate>
 | <in predicate> | <like predicate>
 | <exists predicate> | <null predicate>
 <comparison predicate> ::= <value expression> <comp op> <value expression>
 <comp op> ::= = | < | > | <= | >= | <> | [NOT] ELEMENT OF
 | [NOT] CONTAINS | [NOT] SUBSET OF
 <between predicate> ::= <value expression> [NOT] BETWEEN
 <value expression> AND <value expression>
 <in predicate> ::= <value expression tuple list> IN <nested query expression>
 <value expression tuple list> ::= <value expression>
 | <<value expression> [{, <value expression>}...] >
 <like predicate> ::= <not further defined>
 <exists predicate> ::= EXISTS <nested query expression>
 <null predicate> ::= <column spec> IS [NOT] NULL
 <nested query expression> ::= <table name> | ((query expression))
 <column list> ::= [ALL BUT] <column spec> [{, <column spec>}...]
 <function> ::= MAX | MIN | AVG | SUM | COUNT | DISTINCT | SUBSUME
 <set operator> ::= UNION | DIFFERENCE | INTERSECT
 <data type> ::= <character string type> | <numeric type>
 <value expression> ::= <term> | <value expression> {+|-} <term>
 <term> ::= <factor> ::= | <term> {*/|} <factor>
 <factor> ::= [+|-] <primary>

<primary> ::= **<nested query expression>** | **<value spec>** | **<column spec>**
 | **((value expression))**

<value list> ::= **<value spec>...**

<value spec> ::= **<literal>** | **NULL**

<literal> ::= **<character string literal>** | **<numeric literal>** | **<tuple literal>**
 | **<don't care literal>**

<tuple literal> ::= **<value spec>** [**{**, **<value spec>**]**...**] **>**

<column spec> ::= [**{**(**<reference name>**).**...**]**<column name>**

DML

<dml statement> ::= **<store statement>** | **<modify statement>** | **<erase statement>**

<store statement> ::= **STORE** **<table name>** [**(****<column list>****)**] **{VALUES** **<value list>**
 | **<query expression>**

<modify statement> ::= **MODIFY** **<table name>** [**AS** **<reference name>**]
SET **<set clause>...** [**WHERE** **<search condition>**]

<set clause> ::= **<column name>** = **{****<value expression>** | **(****<dml statement>****)****}**

<erase statement> ::= **ERASE** **<table name>** [**AS** **<reference name>**]
 [**WHERE** **<search condition>**]

DDL

<ddl statement> ::= **<schema>** | **<scheme>**

<schema> ::= **SCHEMA** **{****<table definition>** | **<view definition>****}**...

<table definition> ::= **TABLE** **<table name>** **{****<table element>...** | **<scheme name>****}**

<table element> ::= **<column specification>**
 | **CONSTRAINTS** **<table constraint definition>...**

<column specification> ::= **ITEM** **{****<column definition>** | **(****<table definition>****)****}**

<column definition> ::= <column name> <data type>
 [<column constraint spec>...] [<default clause>]

<column constraint spec> ::= <not null clause> | <unique clause>
 | <references clause> | <check clause>

<not null clause> ::= NOT NULL

<unique clause> ::= UNIQUE

<references clause> ::= REFERENCES <column spec> [<update rule>]
 [<delete rule>]

<check clause> ::= CHECK <search condition>

<default clause> ::= DEFAULT <literal>

<table constraint definition> ::= <unique constraint definition>
 | <referential constraint definition>
 | <check constraint definition>

<unique constraint definition> ::= UNIQUE <column list>

<referential constraint definition> ::= REFERENCES <column list>
 WITH <column list>
 [<update rule>] [<delete rule>]

<update rule> ::= <action> MODIFY

<delete rule> ::= <action> ERASE

<action> ::= CASCADE | NULLIFY | RESTRICT

<check constraint definition> ::= CHECK <search condition> [<defer clause>]

<defer clause> ::= IMMEDIATE | DEFERRED

<view definition> ::= VIEW <table name> AS <query expression>

<scheme> ::= SCHEME <scheme definition>...

<scheme definition> ::= TABLE <scheme name> <table element>...

Bibliography

- [Abi] Abiteboul, S., "Disaggregations in Databases," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta (March 1983), 384-388.
- [AB1] Abiteboul, S. and N. Bidoit, "Non First Normal Form Relations to Represent Hierarchically Organized Data," *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Waterloo (April 1984), 191-200.
- [AB2] Abiteboul, S. and N. Bidoit, "Non First Normal Form Relations: An Algebra Allowing Data Restructuring," *Rapports de Recherche No 347*, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France (November 1984).
- [AG] Abiteboul, S. and G. Grahne, "Update Semantics for Incomplete Databases," *Proceedings of the Eleventh International Conference on Very Large Databases*, Stockholm (August 1985), 1-12.
- [AH] Abiteboul, S. and R. Hull, "Restructuring of Semantic Database Objects and Office Forms," *Extended Abstract*, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France (October 1984).
- [ABU] Aho, A., C. Beeri and J. Ullman, "The Theory of Joins in Relational Databases," *ACM Transactions on Database Systems* 4, 3 (September 1979), 297-314.
- [AMM] Arisawa, H., K. Moriya and T. Miura, "Operations and the Properties on Non-First-Normal-Form Relational Databases," *Proceedings of the Ninth International Conference on Very Large Databases*, Florence (October 1983), 197-204.
- [AC] Arora, A. and C. Carlson, "On the Flexibility Provided by Conflict-Free Normalization," *Proceedings of the 6th International Computer Software Applications Conference*, Chicago (November 1982), 202-206.
- [AM] Atzeni, P. and N. Morfuni, "Functional Dependencies in Relations with Null Values," *Information Processing Letters* 18, 4 (May 1984),

233-238.

- [B+] Bancilhon, F., D. Fortin, S. Gamerman, J. Laubin, P. Richard, M. Scholl, D. Tusera and A. Verroust, "Verso : A Relational Back End Data Base Machine." In *Advanced Database Machine Architecture*, D. Hsiao, Ed., Prentice-Hall, Englewood Cliffs, 1983, 1-18.
- [BaKh] Bancilhon, F. and S. Khoshafian, "A Calculus for Complex Objects," *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Cambridge (March 1986), 53-59.
- [BRS] Bancilhon, F., P. Richard and M. Scholl, "On Line Processing of Compacted Relations," *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City (September 1982), 263-269.
- [Bat] Batory, D., J. Barnett, J. Garza, K. Smith, K. Tsukuda, B. Twichell and T. Wise, "GENESIS: A Reconfigurable Database Management System," TR-86-07, Department of Computer Science, University of Texas at Austin, (March 1986).
- [BaKi] Batory, D. and W. Kim, "Modeling Concepts for VLSI CAD Objects," *ACM Transactions on Database Systems* 10, 3 (September 1985), 322-346.
- [BBG] Beeri, C., P. Bernstein and N. Goodman, "A Sophisticate's Introduction to Database Normalization Theory," *Proceedings of the Fourth International Conference on Very Large Databases*, West-Berlin (September 1978), 113-124.
- [BFH] Beeri, C., R. Fagin, J. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Toronto (August 1977), 47-61.
- [BFMY] Beeri, C., R. Fagin, D. Maier and M. Yannakakis, "On the Desirability of Acyclic Database Schemes," *Journal of the ACM* 30, 3 (July 1983), 479-513.
- [BeK1] Beeri, C. and M. Kifer, "Comprehensive Approach to the Design of Relational Database Schemes," *Proceedings of the Tenth International Conference on Very Large Databases*, Singapore (August

1984), 196-207.

- [BeK2] Beeri, C. and M. Kifer, "Elimination of Intersection Anomalies From Database Schemes," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta (March 1983), 340-351.
- [BV] Beeri, C. and M. Vardi, "A Proof Procedure for Data Dependencies," *Journal of the ACM* 31, 4 (October 1984), 718-741.
- [BR] Benn, W. and B. Radig, "Erweiterte Anfragen nach Relationengebilden in Form nichtnormalisierter Relationen." In *Datenbank-Systeme für Büro, Technik und Wissenschaft*, A. Blaser, P. Pistor, Eds., Informatik-Fachberichte Nr. 94, Springer-Verlag, Berlin, 1985, 487-491.
- [Bis1] Biskup, J., "A Formal Approach to Null Values in Database Relations." In *Advances in Database Theory, Volume 1*, H. Gallaire, J. Minker, J. Nicolas, Eds., Plenum Press, New York, 1981, 299-341.
- [Bis2] Biskup, J., "A Foundation of Codd's Relational Maybe-Operations," *ACM Transactions on Database Systems* 8, 4 (December 1983), 608-636.
- [Bis3] Biskup, J., "Inferences of Multivalued Dependencies in Fixed and Undetermined Universes," *Theoretical Computer Science* 10 (1980), 93-105.
- [Bis4] Biskup, J., "On the Complementation Rule for Multivalued Dependencies in Database Relations," *Acta Informatica* 10 (1978), 297-305.
- [Bra] Bradley, J., "Application of SQL/N to the Attribute-Relation Associations Implicit in Functional Dependencies," *International Journal of Computer and Information Sciences* 12, 2 (1983), 65-86.
- [C+] Chamberlin, D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal of Research and Development* 20, 6 (November 1976), 560-575.
- [Cod1] Codd, E., "A Relational Model for Large Shared Data Banks," *Communications of the ACM* 13, 6 (June 1970), 377-387.

- [Cod2] Codd, E., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems* 4, 4 (December 1979), 397-434.
- [Cod3] Codd, E., "Further Normalization of the Data Base Relational Model." In *Courant Computer Science Symposium 6 on Data Base Systems*, R. Rustin, Ed., Prentice-Hall, New York, 1971, 33-64.
- [Dat1] Date, C., "A Critique of the SQL Database Language," *ACM SIGMOD Record* 14, 3 (November 1984), 8-54.
- [Dat2] Date, C., "Some Principles of Good Language Design with especial reference to the design of database languages," *ACM SIGMOD Record* 14, 3 (November 1984), 1-7.
- [Dat3] Date, C., "The Outer Join," *ICOD-2 Proceedings Second International Conference on Databases*, Cambridge (September 1983), 76-106.
- [Del] Delobel, C., "Normalization and Hierarchical Dependencies in the Relational Data Model," *ACM Transactions on Database Systems* 3, 3 (September 1978), 201-222.
- [DGW] Deppisch, U., J. Günauer and G. Walch, "Speicherungsstrukturen und Adressierungstechniken für Komplexe Objekte des NF²-Relationenmodells." In *Datenbank-Systeme für Büro, Technik und Wissenschaft*, A. Blaser, P. Pistor, Eds., Informatik-Fachberichte Nr. 94, Springer-Verlag, Berlin, 1985, 441-459.
- [D+] Deppisch, U., V. Obermeit, H.-B. Paul, H.-J. Schek, M. Scholl and G. Weikum, "Ein Subsystem zur Stablen Speicherung Versionenbehafteter, Hierarchisch Strukturierter Tupel." In *Datenbank-Systeme für Büro, Technik und Wissenschaft*, A. Blaser, P. Pistor, Eds., Informatik-Fachberichte Nr. 94, Springer-Verlag, Berlin, 1985, 421-440.
- [Eps] Epstein, R., "Techniques for processing of aggregates in relational database systems," ERL/UCB Memo M79/8, Electronics Research Laboratory, University of California, Berkley (February 1979).
- [Fag1] Fagin, R., "A Normal Form for Relational Databases That is Based on Domains and Keys," *ACM Transactions on Database Systems* 6,

- 3 (September 1981), 387-415.
- [Fag2] Fagin, R., "Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systems* 2, 3 (September 1977), 262-278.
- [Fag3] Fagin, R., "Normal forms and relational database operators," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Boston (1979), 153-160.
- [FMU] Fagin, R., A. Mendelzon and J. Ullman, "A simplified universal relation assumption and its properties," *ACM Transactions on Database Systems* 7, 3 (September 1982), 343-360.
- [FSTV] Fischer, P., L. Saxton, S. Thomas and D. Van Gucht "Interactions between Dependencies and Nested Relational Structures," *Journal of Computer and System Sciences* 31, 3 (December 1985).
- [FT] Fischer, P. and S. Thomas, "Operators for Non-First-Normal-Form Relations," *Proceedings of the 7th International Computer Software Applications Conference*, Chicago (November 1983), 464-475.
- [FV1] Fischer, P. and D. Van Gucht, "Determining when a Structure is a Nested Relation," *Proceedings of the Eleventh International Conference on Very Large Databases*, Stockholm (August 1985), 171-180.
- [FV2] Fischer, P. and D. Van Gucht, "Structure of Relations Satisfying Certain Families of Dependencies." In *Proceedings of the 2nd Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 182*, K. Melhorn, Ed., Springer-Verlag, Berlin, 1985, 131-142.
- [FV3] Fischer, P. and D. Van Gucht, "Weak Multivalued Dependencies," *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Waterloo (April 1984), 266-274.
- [Fur] Furtado, A., "Horizontal Decomposition to Improve a Non-BCNF Scheme," *ACM SIGMOD Record* 12, 1 (October 1981), 26-32.
- [FK] Furtado, A. and L. Kerschberg, "An algebra of quotient relations," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Toronto (1977), 1-8.

- [Gol] Goldstein, B., "Constraints on Null Values in Relational Databases," *Proceedings of the Seventh International Conference on Very Large Databases*, Cannes (September 1981), 101-110.
- [Gran] Grant, J., "Null Values in a Relational Data Base," *Information Processing Letters* 6, 5 (October 1977), 156-157.
- [Grah] Grahne, G., "Dependency Satisfaction in Databases with Incomplete Information," *Proceedings of the Tenth International Conference on Very Large Databases*, Singapore (August 1984), 37-45.
- [GR] Grahne, G. and K. Raiha, "Database Decomposition into Fourth Normal Form," *Proceedings of the Ninth International Conference on Very Large Databases*, Florence (October 1983), 186-196.
- [Gray] Gray, M., "Implementing Unknown and Imprecise Values in Databases," *Proceedings of the 1st British National Conference on Databases*, Cambridge (July 1981). Republished in *Databases*, ed. S. Deen and P. Hammersley, John Wiley & Sons, New York, 1981, 146-158.
- [GP] Gründig, L. and P. Pistor, "Land-Information-Systeme und Ihre Anforderungen an Datenbank-Schnittstellen." In *Sprachen für Datenbanken*, J. Schmidt, Ed., Informatik-Fachberichte Nr. 72, Springer-Verlag, Berlin, 1983, 61-75.
- [Han] Hanatani, Y., "Join-Decomposition of MVD-Constraints Using the Characterization by 'Bases'-An Introductory Study," *International Conference on Foundations of Data Organization*, Kyoto, Japan (May 1985), 183-190.
- [HL] Haskin, R. and R. Lorie, "Using a Relational Database System for Circuit Design," *IEEE Database Engineering Bulletin* 5, 2 (June 1982), 10-14.
- [HR] Heiler, S. and A. Rosenthal, "G-WHIZ, a Visual Interface for the Functional Model with Recursion," *Proceedings of the Eleventh International Conference on Very Large Databases*, Stockholm (August 1985), 209-218.
- [Hon] Honeyman, P., "Testing Satisfaction of Functional Dependencies." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.

- [Hsi] Hsieh, Y., "An Unified Query-By-Example Information Manipulation Language Based on Functional Data Model," *Proceedings of the 7th International Computer Software Applications Conference*, Chicago (November 1983), 571-579.
- [HY] Hull, R. and C. Yap, "The Format Model: A Theory of Database Organization," *Journal of the ACM* 31, 3 (July 1984), 518-537.
- [IL1] Imieliński, T. and W. Lipski, "Incomplete Information in and Dependencies in Relational Databases," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, San Jose (May 1983), 178-184.
- [IL2] Imieliński, T. and W. Lipski, "On Representing Incomplete Information in a Relational Database," *Proceedings of the Seventh International Conference on Very Large Databases*, Cannes (September 1981), 388-397.
- [Jac1] Jacobs, B., *Applied Database Logic I: Fundamental Database Issues*, Prentice-Hall, Englewood Cliffs, 1984.
- [Jac2] Jacobs, B., *Applied Database Logic II: Heterogeneous Distributed Query Processing*, Prentice-Hall, Englewood Cliffs, 1985.
- [Jac3] Jacobs, B., "On Database Logic," *Journal of the ACM* 29, 2 (April 1982), 310-332.
- [JW] Jacobs, B. and C. Walczak, "A Generalized Query-by-Example Data Manipulation Language Based on Database Logic," *IEEE Transactions on Software Engineering* 9, 1 (January 1983), 40-56.
- [Jae1] Jaeschke, G., "An Algebra of Power Set Type Relations," TR 82.12.002, Heidelberg Scientific Center, IBM Germany (December 1982).
- [Jae2] Jaeschke, G., "Nonrecursive Algebra for Relations with Relation Valued Attributes," TR 85.03.001, Heidelberg Scientific Center, IBM Germany (March 1985).
- [Jae3] Jaeschke, G., "Recursive Algebra for Relations with Relation Valued Attributes," TR 85.03.002, Heidelberg Scientific Center, IBM Germany (March 1985).

- [JS] Jaeschke, G. and H. Schek, "Remarks on the Algebra of Non First Normal Form Relations," *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles (March 1982), 124-138.
- [Joh] Johnson, R., "Modelling Summary Data," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Ann Arbor (April 1981), 93-97.
- [KF] Kambayashi, Y. and T. Furukawa, "Semantic Constraints Expressed by Network Model," *International Conference on Foundations of Data Organization*, Kyoto, Japan (May 1985), 201-206.
- [KTT] Kambayashi, Y., K. Tanaka and K. Takeda, "Synthesis of Unnormalized Relations Incorporating More Meaning," *Information Sciences* 29 (1983), 201-247.
- [KTTY] Kambayashi, Y., K. Tanaka, K. Takeda and S. Yajima, "Representation of Relations for Database Output Utilizing Data Dependencies," *Proceedings of the 15th Hawaii International Conference on System Sciences* (January 1982), 69-78.
- [KTW] Kappel, G., A. Tjoa and R. Wagner, "Form Flow Systems Based on NF^2 -Relations." In *Datenbank-Systeme für Büro, Technik und Wissenschaft*, A. Blaser, P. Pistor, Eds., Informatik-Fachberichte Nr. 94, Springer-Verlag, Berlin, 1985, 234-252.
- [Kat] Katsuno, H., "An Extension of Conflict-Free Multivalued Dependencies," *ACM Transactions on Database Systems* 9, 2 (June 1984), 309-326.
- [KW] Keller, A. and M. Wilkins, "On the Use of an Extended Relational Model to Handle Changing Incomplete Information," *IEEE Transactions on Software Engineering* 11, 7 (July 1985), 620-633.
- [Klu] Klug, A., "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions," *Journal of the ACM* 29, 3 (July 1982), 699-717.
- [Kob] Kobayashi, I., "An Overview of the Database Management Technology," Technical Report TRCS-4-1, Sanno College, Kanagawa, Japan (June 1980).

- [Kor] Korth, H., "Extending the Scope of Relational Languages," *IEEE Software* 3, 1 (January 1986), 19-28.
- [KS] Korth, H. and A. Silberschatz, "A User-Friendly Operating System Interface Based on the Relational Data Model," *International Symposium on New Directions in Computing*, Trondheim (August 1985), 302-310.
- [K+] Korth, H., G. Kuper, J. Feigenbaum, A. Van Gelder and J. Ullman, "System/U: A Database System Based on the Universal Relation Assumption," *ACM Transactions on Database Systems* 9, 3 (September 1984), 331-347.
- [Kun] Kunii, H., "Graph Data Language: A High Level Access-Path Oriented Language," PhD Dissertation, TR-216, Department of Computer Science, University of Texas at Austin (May 1983).
- [KV1] Kuper, G. and M. Vardi, "A New Approach to Database Logic," *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Waterloo (April 1984), 86-96.
- [KV2] Kuper, G. and M. Vardi, "On the Expressive Power of the Logical Data Model," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Austin (May 1985), 180-187.
- [LaP] LaCroix, M. and A. Pirotte, "Generalized Joins," *ACM SIGMOD Record* 8, 3 (September 1976), 14-15.
- [LeP] LeDoux, C. and D. Parker, "Reflections on Boyce-Codd Normal Form," *Proceedings of the Tenth International Conference on Very Large Databases*, Mexico City (September 1982), 131-141.
- [LST] Lewis, E., L. Sekino and P. Ting, "A Canonical Representation for the Relational Schema and Logical Data Independence," *Proceedings of the 1st International Computer Software Applications Conference*, Chicago (November 1977), 276-280.
- [Lie1] Lien, Y., "Hierarchical Schemata for Relational Databases," *ACM Transactions on Database Systems* 6, 1 (March 1981), 48-69.
- [Lie2] Lien, Y., "Multivalued Dependencies with Null Values in Relational Data Bases," *Proceedings of the Fifth International Conference on*

Very Large Databases, Rio De Janeiro (October 1979), 61-66.

- [Lie3] Lien, Y., "On the Equivalence of Database Models," *Journal of the ACM* 29, 2 (April 1982), 333-362.
- [LTK] Ling, T., F. Tompa and T. Kameda, "An Improved Third Normal Form for Relational Databases," *ACM Transactions on Database Systems* 6, 2 (June 1981), 329-346.
- [Lip1] Lipski, W., "On Databases with Incomplete Information," *Journal of the ACM* 28, 1 (January 1981), 41-70.
- [Lip2] Lipski, W., "On Semantic Issues Connected with Incomplete Information Databases," *ACM Transactions on Database Systems* 4, 3 (September 1979), 262-296.
- [Lor] Lorie, R., "Issues in Databases for Design Applications," IBM Research Report RJ3176, July 1981.
- [L+] Lum, V., P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner and J. Woodfill, "Design of an Integrated DBMS to Support Advanced Applications," *International Conference on Foundations of Data Organization*, Kyoto, Japan (May 1985), 21-31.
- [Mac] Macleod, I., "SEQUEL as a Language for Document Retrieval," *Journal of the American Society for Information Science* (September 1979), 243-249.
- [Mai1] Maier, D., "Discarding the Universal Relation Instance Assumption: Preliminary Results." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.
- [Mai2] Maier, D., *Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983.
- [Mak] Makinouchi, A., "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model," *Proceedings of the Third International Conference on Very Large Databases*, Tokyo (October 1977), 447-453.
- [ML] Meier, A. and R. Lorie, "Implicit Hierarchical Joins for Complex Objects," IBM Research Report RJ3775, February 1983.

- [Men1] Mendelzon, A., "Functional Dependencies in Logic Programs," *Proceedings of the Eleventh International Conference on Very Large Databases*, Stockholm (August 1985), 324-330.
- [Men2] Mendelzon, A., "On Axiomatizing Multivalued Dependencies in Relational Databases," *Journal of the ACM* 26, 1 (January 1979), 37-44.
- [Orm] Orman, L., "Semantics of indexed sets," Cornell University BPA Working Paper (1981).
- [Osb] Osborn, S., "Relational Databases with Partially Formatted Records," *Proceedings of the 6th International Computer Software Applications Conference*, Chicago (November 1982), 589-593.
- [Ozs] Özsoyoğlu, Z., Personal Communication, December, 1985.
- [OMO] Özsoyoğlu, G., V. Matos and Z. Özsoyoğlu, "Query Processing Techniques in the Summary-Table-by-Example Database Query Language," Technical Report, Computer Engineering and Science Department, Case Western Reserve University (1985).
- [OO1] Özsoyoğlu, G. and Z. Özsoyoğlu, "An Extension of Relational Algebra for Summary Tables," *Proceedings of the 2nd International (LBL) Conference on Statistical Database Management*, Los Angeles (September 1983), 202-211.
- [OO2] Özsoyoğlu, G. and Z. Özsoyoğlu, "SSDB—An Architecture for Statistical Databases," *Proceedings of the 4th Jerusalem Conference on Information Technology*, Jerusalem (May 1984), 327-341.
- [OOM1] Özsoyoğlu, G., Z. Özsoyoğlu and V. Matos, "A Language and a Physical Organization Technique for Summary Tables," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Austin (May 1985), 3-16.
- [OOM2] Özsoyoğlu, G., Z. Özsoyoğlu and V. Matos, "Extending Relational Algebra and Relational Calculus with Aggregate Functions and Set-Valued Attributes," Technical Report, Computer Engineering and Science Department, Case Western Reserve University (1985).
- [OO3] Özsoyoğlu, Z. and G. Özsoyoğlu, "A Query Language for Statistical Databases." In *Query Processing in Database Systems*, W. Kim, D.

- Reiner, D. Batory, Eds., Springer-Verlag, Berlin, 1985, 171-187.
- [OY1] Özsoyoğlu, Z. and L. Yuan, "A Normal Form for Nested Relations," *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Portland (March 1985), 251-260.
- [OY2] Özsoyoğlu, Z. and L. Yuan, "Reduced MVD's and Minimum Covers," Technical Report CES-84-06, Computer Engineering and Science Department, Case Western Reserve University (1984).
- [PP] Parker, D. and K. Parsaye-Ghomi, "Inferences Involving Embedded Multivalued Dependencies and Transitive Dependencies," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Santa Monica (May 1980), 52-57.
- [PHH] Pistor, P., B. Hansen and M. Hansen, "Eine sequenzielle Sprachschnittstelle für das NF2-Modell." In *Sprachen für Datenbanken*, J. Schmidt, Ed., Informatik-Fachberichte Nr. 72, Springer-Verlag, Berlin, 1983, 134-147.
- [PS] Pistor, P. and H.-J. Schek, "On an Integrated System for the Management of Formatted and Unformatted Data," Technical Report 82.01.001, Heidelberg Scientific Center, IBM Germany (January 1981).
- [PT] Pistor, P. and R. Traummüller, "A Data Base Language for Sets, Lists, and Tables," Technical Report 85.10.004, Heidelberg Scientific Center, IBM Germany (October 1985).
- [RKB] Roth, M., H. Korth and D. Batory, "SQL/NF: A Query Language for \neg 1NF Relational Databases," TR-85-19, Department of Computer Science, University of Texas at Austin (September 1985).
- [RKS1] Roth, M., H. Korth and A. Silberschatz, "Extended Algebra and Calculus for \neg 1NF Relational Databases," TR-84-36, Department of Computer Science, University of Texas at Austin (December 1984), revised January 1986.
- [RKS2] Roth, M., H. Korth and A. Silberschatz, "Null Values in \neg 1NF Relational Databases," TR-85-32, Department of Computer Science, University of Texas at Austin (December 1985).

- [Sac] Sacca, D., "On the Recognition of Coverings of Acyclic Database Hypergraphs," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta (March 1983), 297-304.
- [Sad] Sadri, F., "A Normal Form with respect to GFD's and TD's and New Conditions for Decomposition of Relational Databases," Technical Report, Princeton University, 1981.
- [SU1] Sadri, F. and J. Ullman, "Template Dependencies: A Large Class of Dependencies in Relational Databases and its Complete Axiomatization," *Journal of the ACM* 29, 2 (April 1982), 363-372.
- [SU2] Sadri, F. and J. Ullman, "The Interaction between Functional Dependencies and Template Dependencies," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Santa Monica (May 1980), 45-51.
- [Sch1] Schek, H.-J., "Methods for the administration of textual data in database systems." In *Information Retrieval Research*, Oddy, Robinson, Van Rijsbergen and Williams, Eds., Butterworth, London, 1981, 218-235.
- [Sch2] Schek, H.-J., "Towards a Basic Relational NF^2 Algebra Processor," *International Conference on Foundations of Data Organization*, Kyoto, Japan (May 1985), 173-182.
- [SP] Schek, H.-J. and P. Pistor, "Data Structures for an Integrated Data Base Management and Information Retrieval System," *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City (September 1982), 197-207.
- [ScS1] Schek, H.-J. and M. Scholl, "An Algebra for the Relational Model with Relation-Valued Attributes," TR DVSI-1984-T1, Technical University of Darmstadt, Darmstadt, West Germany (1984).
- [ScS2] Schek, H.-J. and M. Scholl, "Die NF^2 -Relationenalgebra zur Einheitlichen Manipulation Externer, Konzeptueller und Interner Datenstrukturen." In *Sprachen für Datenbanken*, J. Schmidt, Ed., Informatik-Fachberichte Nr. 72, Springer-Verlag, Berlin, 1983, 113-133.

- [Sci1] Sciore, E., "Improving Database Schemes by Adding Attributes," *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, (March 1983), 379-383.
- [Sci2] Sciore, E., "Null Values, Updates, and Normalization in Relational Databases," Technical Report, Department of Electrical Engineering and Computer Science, Princeton University (December 1979).
- [Sci3] Sciore, E., "Real-World MVD's," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Ann Arbor (April 1981), 121-132.
- [Sci4] Sciore, E., "Some Observations on Real-World Data Dependencies." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.
- [Shi] Shipman, D., "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems* 6, 1 (March 1981), 140-173.
- [SLTC] Shu, N., Y. Lum, F. Tung and C. Chang, "Specification of Forms Processing and Business Procedures for Office Automation," *IEEE Transactions on Software Engineering* 8, 5 (September 1982), 499-512.
- [SmS] Smith, J. and D. Smith, "Database abstractions: aggregation and generalization," *ACM Transactions on Database Systems* 2, 2 (June 1977), 105-133.
- [TKY] Tanaka, K., Y. Kambayashi and S. Yajima, "Properties of Embedded Multivalued Dependencies in Relational Databases," *The Transactions of the IECE of Japan* 62, 8 (August 1979), 536-543.
- [Tom] Tompa, F., "A Practical Example of the Specification of Abstract Data Types," *Acta Informatica* 13 (1980), 205-224.
- [Tho] Thomas, S., "A Non-First-Normal-Form Relational Database Model," PhD Dissertation, Vanderbilt University (1983).
- [TF] Thomas, S. and P. Fischer, "Nested Relational Structures." In *The Theory of Databases*, P. Kanellakis, Ed., JAI Press, 1986.

- [Ull] Ullman, J., *Principles of Database Systems*, 2nd Edition, Computer Science Press, Potomac, MD, 1982.
- [Van] Van Gucht, D., "Theory of Unnormalized Relational Structures," PhD Dissertation, Vanderbilt University (1985).
- [VF] Van Gucht, D. and P. Fischer, "Some Classes of Multilevel Relational Structures," *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Cambridge (March 1986), 60-69.
- [Vas1] Vassiliou, Y., "Functional Dependencies and Incomplete Information," *Proceedings of the Sixth International Conference on Very Large Databases*, Montreal (October 1980), 260-269.
- [Vas2] Vassiliou, Y., "Null Values in Data Base Management: A Denotational Semantics Approach," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Boston (1979), 162-169.
- [Vas3] Vassiliou, Y., "Testing Satisfaction of FDs on a multi-relation database "fast"." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.
- [Wal] Walker, A., "Time and Space in a Lattice of Universal Relations with Blank Entries." In *Proceedings of the XP1 Workshop on Relational Database Theory*, New York, June 1980.
- [Won] Wong, E., "A Statistical Approach to Incomplete Information in Database Systems," *ACM Transactions on Database Systems* 7, 3 (September 1982), 470-488.
- [X3H2] X3H2-84-2, "(Draft Proposed) Relational Database Language," American National Standards Committee on Computer and Information Processing, January, 1984.
- [YO] Yuan, L. and Z. Özsoyoğlu, "Unifying Functional and Multivalued Dependencies for Relational Database Design," *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Cambridge (March 1986), 183-190.
- [Zan1] Zaniolo, C., "A Formal Treatment of Nonexistent Values in Database Relations," Technical Report, Bell Laboratories, Holmdel, NJ

(January 1983).

- [Zan2] Zaniolo, C., "Database Relations with Null Values," *Journal of Computer and System Sciences* 28, 1 (February 1984) 142-166.
- [Zan3] Zaniolo, C., "Design of Relational Views Over Network Schemes," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Boston (May 1979), 179-190.
- [Zan4] Zaniolo, C., "Relational Views in a Data Base System Support for Queries," *Proceedings of the 1st International Computer Software Applications Conference*, Chicago (November 1977), 267-275.
- [Zan5] Zaniolo, C., "The Database Language GEM," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, San Jose (May 1983), 207-218.
- [ZM] Zaniolo, C. and M. Melkanoff, "On the Design of Relational Database Schemata," *ACM Transactions on Database Systems* 6, 1 (March 1981), 1-47.
- [Zlo] Zloof, M., "Query-By-Example: Operations on the Transitive Closure," RC-5526, IBM Thomas J. Watson Research Center, Yorktown Heights, New York (October 1976).

Vita

PII Redacted

Mark Aron Roth [REDACTED]

the son of Russell W. Roth and Felizitas P. Roth. After completing his work at Fenton High School, Bensenville, Illinois, in 1975, he entered Illinois Institute of Technology in Chicago, Illinois. In May, 1978, he received the degree of Bachelor of Science with High Honors and was commissioned as a Second Lieutenant in the United States Air Force. He was selected to attend the Air Force Institute of Technology at Wright-Patterson Air Force Base, Ohio and in December, 1979, he received the degree of Master of Science. During the following years he was employed by the United States Air Force as a computer systems analyst and programmer in support of computer wargaming for the Air Force's professional military education schools. In June, 1980, he was promoted to Captain and in August, 1983, he was awarded the Meritorious Service Medal. In August, 1983, he entered The Graduate School of The University of Texas at Austin.

[REDACTED]

This dissertation was typed by the author.