NULL VALUES IN ¬1NF RELATIONAL DATABASES

Mark A. Roth, Henry F. Korth &
Abraham Silberschatz

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

TR-85-32 December 1985

Null Values in $\neg$1NF Relational Databases

Mark A. Roth
Henry F. Korth [†]
Abraham Silberschatz [‡]

Department of Computer Science
University of Texas at Austin
Austin, Texas 78712

Abstract

The desire to extend the applicability of the relational model beyond traditional data-processing applications has stimulated interest in non-first-normal-form relations in which the attributes of a relation can take on values which are sets or even relations themselves. In this paper, we study the role of null values in the non-first-normal-form relational model. We extend the traditional theory and study the properties of extended non-first-normal-form relational operators operating on relations containing nulls. The no-information, unknown, and non-existent interpretation of nulls are discussed and the meaning of "empty set" is clarified. Finally, contrary to several previous results, we determine that the traditional axiomatization of functional and multivalued dependencies is valid in the presence of nulls.

1. Introduction

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, computer-aided design, and text retrieval systems are a few examples of non-traditional applications that require specialized database support. One of the stumbling blocks in using traditional relational databases and relational theory 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. For this reason, non-first-normal-form ($\neg$1NF) relations were proposed in which the attributes of a relation can take on values which are sets or even relations themselves. This created a need to reexamine the fundamentals of relational database theory in light of this new assumption and opened the door for the introduction of operators which take advantage of the *nested* structure of $\neg$1NF relations.

To illustrate this, consider an employee relation which is in 1NF (Figure 1-1a), and a possible $\neg$1NF structuring of it (Figure 1-1b). The $\neg$1NF relation has two tuples, ⟨Smith, {⟨Sam⟩, ⟨Sue⟩}, {⟨typing⟩, ⟨filing⟩}⟩ and ⟨Jones, {⟨Joe⟩, ⟨Mike⟩}, {⟨typing⟩, ⟨dictation⟩, ⟨data entry⟩}⟩. 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.

One additional advantage of using a $\neg$1NF structuring of the database is that fewer relations are needed to maintain normal forms like third (3NF) and fourth normal form (4NF). To illustrate this, consider that in the employee example of Figure 1-1a, we would expect the multivalued dependency, employee $\rightarrow\!\!\!\rightarrow$ child | skill, to hold. In order to reduce redundancy and avoid update anomalies we would, according to the given dependency, decompose this relation into its projections (employee, child) and (employee, skill). The result is a 4NF database scheme as shown in Figure 1-2. To view the entire database we must use a join operator to reassemble the original relation. With a straightforward carryover of the definitions of dependencies and normal forms, the $\neg$1NF relation of Figure 1-2b is in 4NF, and does not need to be decomposed. Queries involving child and skill will be simpler and more efficient since a join of decomposed relations is not necessary.

† Research partially supported by an IBM Faculty Development Award and NSF grant DCR-8507224
‡ Research partially supported by NSF grants DCR-8507224, MCS81-22039, MCS82-14613, and MCS81-04017

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) ¬1NF.

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. 4NF decomposition of employee relation.

A problem arises in the ¬1NF representation of the database. If we have an employee with several skills and no children, then, in the database of Figure 1-2, we simply add tuples to the decomposed (employee, skill) relation and add nothing to the (employee, child) relation. Now, consider the representation of this information in the ¬1NF relation of Figure 1-1b. 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 ¬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 paper 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 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 ¬1NF databases. We still want the advantages of handling null values in a standard and unambiguous system. However, the need for nulls is even more critical in a ¬1NF database since otherwise we lose some of its advantages.

The remainder of this paper is organized as follows. In section 3, 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 above assumptions. These extensions serve as a basis for the main results of this paper, the extension to $\neg$1NF. In section 4, we define the $\neg$1NF model we will be using. Two new operators used to restructure relations, *nest* and *unnest*, are defined and *partitioned normal form* is presented as a desirable goal in structuring $\neg$1NF relations. We provide extensions to the traditional relational operators which work with $\neg$1NF relations and maintain partitioned normal form. In section 5, we extend the null value theory presented in section 3 to $\neg$1NF relations, and further extend the operators of section 4 to deal with null values. Finally, in section 6, we discuss dependency theory, shedding some new light on the problem of nulls when dealing with functional and multivalued dependencies, and their axiomatization.

2. Notation

We will assume, without loss of generality, that all attributes of our relations are contained in a finite universe of attributes, U. Each attribute $A \in U$ may assume values drawn from a domain, DOM(A). 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* $R = (A_1, A_2, \ldots, A_n)$ where $A_i \in U$, $1 \leq i \leq n$. The set of attributes in a relation scheme rule R are denoted E_R. For $A \in E_R$, an A-value is an assignment of a value from 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 defined on scheme R is a set of E_R-values, with the elements of this set called tuples of r. We will generally use upper case letters from the beginning of the alphabet to represent single attributes and upper case letters from the end of the alphabet to represent sets of attributes. We also let XY denote $X \cup Y$.

The operators $\cup$, $\cap$, $-$, $\times$, $\bowtie$, π, and σ represent the standard relational operators on 1NF relations without null 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.

As we will occasionally refer to various data dependencies we provide reference definitions here.

- 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 $t_1[Y] = t_2[Y]$.
- Multivalued dependency (MVD). Let r be a relation on scheme R, with X and Y subsets of E_R and $Z = E_R - XY$. Relation r *satisfies* the *multivalued dependency* $X \rightarrow\rightarrow Y$ 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 $t_3[Z] = t_2[Z]$.
- Join dependency (JD). Let r be a relation on scheme R, with $X_1, X_2, \ldots, X_n$ subsets of E_R. Relation r *satisfies* the *join dependency* $*[X_1, X_2, \ldots, X_n]$ if r decomposes losslessly onto $X_1, X_2, \ldots, X_n$. That is,

$$r = \pi_{X_1}(r) \bowtie \pi_{X_2}(r) \bowtie \cdots \bowtie \pi_{X_n}(r).$$

Fixing a particular scheme, the set of all relations on that scheme that satisfies a set of dependencies D is denoted $SAT(D)$.

3. 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, Cod, Gran, Mai1], and *nonexistent* by [Lie1, Lie2, 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 ¬1NF relations.

Other proposals for nulls are rather sophisticated, involving partial specification [Lip1, Lip2, IL1, IL2], probability distributions [Won], and conditional tuples [KW], but it could be argued "that the complexity of their management is not justified by their richer semantics" [AM; 233].

3.1 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 3.1: Let $\{d_1, d_2, \ldots, d_n\}$ be a domain and $D = \{d_1, d_2, \ldots, d_n, \mathbf{unk}, \mathbf{dne}, \mathbf{ni}\}$ the corresponding extended domain. A *greatest lower bound* function, $glb(a, b)$, between two values a and b from D is defined in Figure 3-1.

$b \backslash a$	d_1	d_2	$\cdots$	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 3-1. 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 $d_1, d_2, \ldots, d_n$ as more informative than **unk**. See Figure 3-2.

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 6.

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

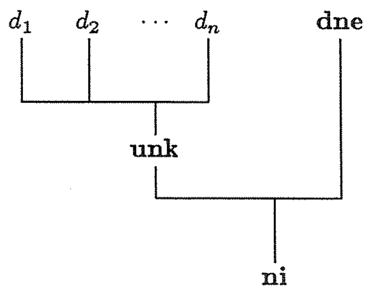


Figure 3-2. Information lattice.

Definition 3.2: 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 = (employee, skill), and R_2 = (employee, child, skill).

Example 3.1: Let

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

denote E_{R_2}-values, and let

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

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 $t_2 \cong t_3$. □

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 3.3: 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 $A \in X \cap Y$, $t[A] = glb(t_1[A], t_2[A])$, and for each attribute $B \notin X \cap Y$, $t[B] = \mathbf{ni}$.

Example 3.2: Using the tuples defined in Example 3.1 we find that

$$t_1 \wedge t_3 = t_2$$
$$t_1 \wedge t_4 = \langle \text{Smith}, \mathbf{ni}, \text{typing} \rangle$$

□

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

Definition 3.4: 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 $t_1, t_2, \ldots, t_n$, we can eliminate tuples in which all attributes have value **ni** (the *null tuple*) [1], 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

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

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

Definition 3.5: 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 3.6: 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 minimum relation in which no tuples in the relation are subsumed by a tuple in the same relation.

Definition 3.7: A relation r constitutes a *minimum representation* for a relation q when no proper subset of r subsumes q.

3.2 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.

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

Definition 3.8: 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 3.9: 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 $SAT(*[R, Q])$. But, for some relation $p \in Rel\!\uparrow\!(RQ)$, $POSS(p)$ is not a subset of $SAT(*[R, Q])$. 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 3.10: 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 $POSS(\gamma'(r)) \supsetneq POSS(p) \supseteq \gamma(POSS(r))$.
2. when γ and γ' are binary operators, for every $r, q \in Rel\!\uparrow$, there is no p in $Rel\!\uparrow$ such that $POSS(r \;\gamma'\; q) \supsetneq POSS(p) \supseteq POSS(r) \;\gamma\; POSS(q)$.

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.

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. These operators, together with attribute renaming, will form a complete set of relational algebra operators for 1NF relations with null values. Some sample relations are shown in Figure 3-3. These will be used to illustrate the new operators.

3.2.1 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. $p = r \cup' q = \widehat{\{t \mid t \mathrel{\tilde{\in}} r \text{ or } t \mathrel{\tilde{\in}} q\}} = \widehat{\{t \mid t \in r \text{ or } t \in q\}}$.

Some examples of null-union are shown in Figure 3-4.

r_1

employee	skill
Smith	typing
Jones	filing
Jones	typing
Adams	**ni**
ni	dictation

r_2

employee	skill
Smith	**ni**
Jones	typing
ni	clerk
unk	dictation

r_3

employee	child	skill
Smith	Sam	typing
Smith	Sue	typing
Jones	**unk**	**ni**

r_4

employee	child
Smith	Sam
Smith	Sue
Jones	**ni**
unk	Joe

Figure 3-3. Some sample relations.

$r_1 \cup' r_2$

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

$r_1 \cup' r_3$

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 3-4. Examples of null-union.

Proposition 3.1: 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 operation. 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. □

Proposition 3.2: 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 $\widehat{p} \in POSS(r) \cup POSS(q)$. There must be $\widehat{r} \in POSS(r)$ and $\widehat{q} \in POSS(q)$ such that $\widehat{p} = \widehat{r} \cup \widehat{q}$. 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 $t_{\widehat{p}} \in \widehat{r}$, and hence in $\widehat{p}$, such that $t_{\widehat{p}} \geq t_p$. A similar argument holds if $t_p \in q$. We conclude $\widehat{p} \geq p$ and so $\widehat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq POSS(r) \cup POSS(q)$.
- $\subseteq$ Let $\widehat{p} \in POSS(p)$. Since $p \geq r$, $\widehat{p} \geq r$ and so $\widehat{p} \in POSS(r)$. Similarly, $\widehat{p} \in POSS(q)$. Therefore, $\widehat{p} \in POSS(r) \cup POSS(q)$, and so $POSS(p) \subseteq POSS(r) \cup POSS(q)$.

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

3.2.2 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

$$p = r -' q = \widehat{\{t \mid t \mathrel{\widetilde{\in}} r \text{ and } t \mathrel{\widetilde{\notin}} q\}} = \{t \mid t \in r \text{ and } \forall s \in q : \neg(s \geq t)\}.$$

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 $\widetilde{\in}$ 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.

$r_1 -' r_2$

employee	skill
Smith	typing
Jones	filing
Adams	**ni**

$r_2 -' r_1$

employee	skill
unk	dictation
ni	clerk

Figure 3-5. Examples of null-difference.

Some examples of null-difference are shown in Figure 3-5. Null-difference is a faithful, and restricted generalization of standard difference. To show that null-difference is not adequate, consider a relation r with one non-null tuple and an empty relation q. Every relation in $POSS(r -' q)$ must subsume r, whereas $POSS(r) - POSS(q)$ contains the empty relation. Thus, $POSS(r -' q) \not\supseteq POSS(r) - POSS(q)$.

Proposition 3.3: The operator null-difference is faithful to standard 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. □

Proposition 3.4: The operator null-difference is a restricted generalization of standard difference with respect to possibility function $POSS$.
Proof: We show that 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, $POSS(p) \not\supseteq POSS(r) - POSS(q)$, which is a contradiction. We conclude that null-difference is a restricted generalization of standard difference for $POSS$. □

We note that a generalized null-intersection operator is not derivable from null-difference alone. Figure 3-6 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 3-6. 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 note also that null-intersection is an adequate and restricted generalization of standard intersection for $POSS$.

3.2.3 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:

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

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 3-6. Examples of null-intersection.

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

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

3.2.4 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 3.5: 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. □

Proposition 3.6: 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 and 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 $POSS$. □

$\pi'_{skill}(r_1)$

skill
dictation
filing
typing

$\pi'_{child,skill}(r_1 \cup' r_3)$

child	skill
Sam	typing
Sue	typing
ni	filing
ni	dictation

Figure 3-7. Examples of null-project.

3.2.5 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 $A_1, A_2, \ldots, A_n \in E_R$. Null-project is defined as follows:

$$\pi'_{A_1,A_2,\ldots,A_n}(r) = \widehat{\{t[A_1A_2\cdots A_n] \mid t \in r\}}$$

Examples of null-project are shown in Figure 3-7.

Proposition 3.7: The operator null-project is faithful to standard project.
Proof: As in the proof of proposition 3.1, 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. □

Proposition 3.8: 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 $\widehat{p} \in \pi_X(POSS(r))$. There must be $\widehat{r} \in POSS(r)$ such that $\widehat{p} = \pi_X(\widehat{r})$. Let t_p be a tuple in p. Then, $t_p \in r[X]$. If $t_p \in r[X]$, there is a tuple $t_{\widehat{p}} \in \widehat{r}[X]$, and hence in $\widehat{p}$, such that $t_{\widehat{p}} \geq t_p$. We conclude $\widehat{p} \geq p$ and so $\widehat{p} \in POSS(p)$. Therefore, $POSS(p) \supseteq \pi_X(POSS(r))$.
$\subseteq$ Let $\widehat{p} \in POSS(p)$. 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 $\widehat{p}$. 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, $\widehat{p} \in \pi_X(POSS(r))$, and so $POSS(p) \subseteq \pi_X(POSS(r))$.
We conclude that null-project is a precise generalization of standard project for $POSS$. □

3.2.6 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,

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

As in [Zan2, LP], 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

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

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

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 3-8. Information preserving equijoin.

Figure 3-8 shows an example of the information-preserving equijoin of r_2 and r_4.

4. The ¬1NF Relational Model

Various researchers have studied the effect of dropping the assumption that all relations be in first-normal-form (1NF). Early work was done by Makinouchi [Mak] and led to the concept of nesting. This was later studied by Jaeschke and Schek [JS] for one level nesting over single attributes and by Thomas and Fischer [TF] in a more general setting. Utilizing ¬1NF relations for structuring database outputs was discussed by Kambayashi, et al. [KTT], while Fischer and Van Gucht [FV1, FV2] looked at dependencies which characterize ¬1NF relations.

Özsoyoğlu and Özsoyoğlu [OO] consider operations similar to that of [JS], and extend the basic algebra for relations by aggregate operators. Özsoyoğlu and Yuan [OY] introduce *nested normal form* for ¬1NF relations. Given a set of attributes U, and a set of MVDs M over U, they give an algorithm to obtain a nested normal form decomposition of U with respect to M, which explicitly represents a set of full and embedded MVDs implied by M and is a faithful and nonredundant representation of U.

Our previous work [RKS] defines a relational calculus and relational algebra for ¬1NF relations and proves their equivalence. We also introduced *partitioned normal form* for ¬1NF relations (described later) which is equivalent to *scheme trees* of [OY] and *formats* of [AB]. Abiteboul and Bidoit [AB] also define some extended operators which are refined in [RKS], where it was also proved that the set of relations in partitioned normal form are closed under the extended operators. Others [Jae1, Jae2, PHH, RKB, Sch2, SS1, SS2] have been developing languages and implementations for ¬1NF relational databases.

4.1 Basic Concepts

A *database scheme* S is a collection of rules of the form $R_j = (R_{j_1}, R_{j_2}, \ldots, R_{j_n})$. The objects R_j, R_{j_i}, $1 \le i \le n$, 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*. The names on the right hand side of rule R_j form a set denoted 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 we require that no two rules can have the same name on the left hand side.

Example 4.1: Consider a slightly expanded version of the employee example used in the introduction. The scheme is

Emp = (employee, Children, Skills),

Children = (name, dob),

Skills = (type, Exams),

Exams = (year, city).

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 4-1. A sample relation on the Emp scheme.

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 4-1. □

In this example the higher order attributes are Emp, Children, Skills and Exams. All others are zero order attributes. We will generally use capitalized names for higher order attributes and uncapitalized names for zero order attributes.

A 1NF database scheme is a collection of rules of the form $R_j = (R_{j_1}, R_{j_2}, \ldots, R_{j_n})$ where all the R_{j_i} are zero order. $\neg$1NF schemes 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, a nested relation is represented simply as a higher order attribute on the right hand side of a rule.

In the relation of Figure 4-1, we would not expect two tuples with employee = 'Smith' since all of Smith's children and skills should be grouped into one tuple. That is, there is no significance in separate groups of children or skills. This led us to restrict the set of $\neg$1NF relations to those that are in *partitioned normal form* (PNF).

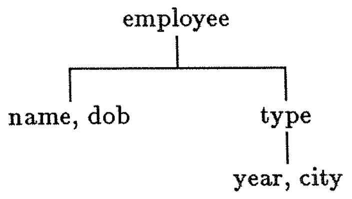
A relation is in PNF if the zero order attributes in the relation form a key for the relation, and each nested relation is also in PNF, that is, the zero order attributes of each nested relation form a key for that relation. The employee relation in Figure 4-1 is in PNF. Note that *employee* is a key for the sample relation, *type* is a key for each *Skills* relation, and (*year*, *city*) is a key for each *Exams* relation. The formal definition is as follows.

Definition 4.1: [RKS] Let r be a relation on scheme R with attributes E_R containing zero order attributes $A_1, A_2, \ldots, A_k$ and higher order attributes $X_1, X_2, \ldots, X_\ell$. Relation r is in *partitioned normal form* (PNF) if and only if the following two conditions hold:

(a) $A_1 A_2 \cdots A_k \rightarrow E_R$.

(b) For all $t \in r$ and for all $X_i : 1 \leq i \leq \ell : \mathcal{R}_{ti}$ is in PNF, where $\mathcal{R}_{ti}$ is the nested relation $t[X_i]$ on scheme X_i.

PNF is precisely the same as structuring $\neg$1NF relations in the form of *scheme trees* [OY] or as *Verso instances* over a *format* [AB]. 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 multivalued dependencies (MVDs) between the attributes in the vertices of the tree. These MVDs allow a 1NF relation to be represented as a $\neg$1NF relation in PNF. The scheme tree and associated MVDs for the Emp scheme are shown in Figure 4-2. 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 $f_1, f_2, \ldots, f_n$ some formats over $Y_1, Y_2, \ldots, Y_n$, resp., such that the sets $X, Y_1, Y_2, \ldots, Y_n$,



employee $\twoheadrightarrow$ name, dob
employee $\twoheadrightarrow$ type, year, city
employee, type $\twoheadrightarrow$ year, city

Figure 4-2. Scheme tree and implied MVDs for employee database.

are pairwise disjoint, then the string $X(f_1)^*(f_2)^* \cdots (f_n)^*$ is a format over the set $XY_1Y_2 \cdots Y_n$. The format for the Emp scheme is employee(name dob)*(type(year city)*)*.

4.2 Operators

In this section, we discuss operators for $\neg$1NF relations without null values. The traditional relational algebra operators can be used with $\neg$1NF relations with only minor modifications to define equality between nested relations and to include more complex renaming rules for cartesian product. With the addition of *unnest* and *nest* operators, this algebra is as powerful as the standard relational algebra working on 1NF relations. We state the formal definitions of *nest* and *unnest* from [RKS, TF].

Definition 4.2: Let R be a relation scheme in database S. Let $\{B_1, B_2, \ldots, B_m\} \subset E_R$ and $C = E_R - \{B_1, B_2, \ldots, B_m\}$. Assume that either the rule $B = (B_1, B_2, \ldots, B_m)$ is in S or that B does not appear on the left hand side of any rule in S. Then the *nest* operator $\nu_{B=(B_1,B_2,\ldots,B_m)}(r)$ produces a relation r' on scheme R' where:

1. $R' = (C, (B_1, B_2, \ldots, B_m)) = (C, B)$ and the rule $B = (B_1, B_2, \ldots, B_m)$ is appended to the set of rules in S if it is not already in S, and
2. $r' = \{t \mid$ there exists a tuple $u \in r$ such that $t[C] = u[C] \wedge t[B] = \{v[B_1B_2 \cdots B_m] \mid v \in r \wedge v[C] = t[C]\}\}$.

Definition 4.3: Let R be a relation scheme in database S. Assume B is some higher order attribute in E_R with an associated rule $B = (B_1, B_2, \ldots, B_m)$ in S. Let $C = E_R - B$. Then the *unnest* operator $\mu_{B=(B_1,B_2,\ldots,B_m)}(r)$ produces a relation r' on scheme R' where:

1. $R' = (C, B_1, B_2, \ldots, B_m)$ and the rule $B = (B_1, B_2, \ldots, B_m)$ is removed from the set of rules in S if it does not appear in any other relation scheme, and
2. $r' = \{t \mid$ there exists a tuple $u \in r$ such that $t[C] = u[C] \wedge t[B_1B_2 \cdots B_m] \in u[B]\}$.

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.

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 4.2: Consider $\neg$1NF relations r_1 and r_2 of Figure 4-3 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. $\Box$

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 similar to the

r_1

A	B*
	B
a	b
	b′
a′	b
	b′

r_2

A	B*
	B
a	b
	b′
a′	b
	b″

$r_1 \cap r_2$

A	B*
	B
a	b
	b′

s_1

A	B
a	b
a	b′
a′	b
a′	b′

s_2

A	B
a	b
a	b′
a′	b
a′	b″

$s_1 \cap s_2$

A	B
a	b
a	b′
a′	b

Figure 4-3. Intersection applied to $\neg$1NF and 1NF relations.

way we defined it for null values. The parallel definitions substitute the set of $\neg$1NF relations for the set of relations with null values, and the unnesting operator for the possibility function.

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, $Rel \cap Rel^* = \emptyset$.

Definition 4.4: 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 4.5: 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. In [RKS], we defined some extended operators in order to work within the domain of PNF relations. We now discuss these extensions in light of the above requirements.

4.2.1 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 4.6: 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 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.

r_1

A	B*	C*
	B	C
a	b	c
		c′

r_2

A	B*	C*
	B	C
a	b′	c

$r_1 \cup^e r_2$

A	B*	C*
	B	C
a	b	c
	b′	c′

$\mu^*(r_1 \cup^e r_2)$

A	B	C
a	b	c
a	b	c′
a	b′	c
a	b′	c′

$\mu^*(r_1)$

A	B	C
a	b	c
a	b	c′

$\mu^*(r_2)$

A	B	C
a	b′	c

$\mu^*(r_1) \cup \mu^*(r_2)$

A	B	C
a	b	c
a	b	c′
a	b′	c

Figure 4-4. Counterexample to preciseness of $\cup^e$.

Proposition 4.1: 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 4.2: Extended union is *not* a *precise* generalization of standard union with respect to unnesting.

Proof: Figure 4-4 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.

As in [OY], 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, and $D(v)$ be the union of all descendants of v, including v. Then the MVD represented by the edge e is $A(u) \rightarrow\rightarrow D(v)$. Also, let $MVD(T)$ be the set of MVD's represented by the edges of T.

Definition 4.7: [OY] Let T be a scheme tree, and $u_1, u_2, \ldots, u_n$ be all the leaf nodes of T. Then the *path set* of T, denoted $P(T)$, is $\{A(u_1), A(u_2), \ldots, A(u_n)\}$. Note that, for a leaf node u, $A(u)$ is the union of all the nodes in the path from the root of T to u in T.

The following proposition gives some properties of a scheme tree.

Proposition 4.3: [OY] If T is a scheme tree, then $MVD(T) \Longleftrightarrow *(P(T))$. □

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 4-4 the MVD A $\rightarrow\rightarrow$ B | 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. We then perform a standard union among the corresponding decomposed relations, and finally rejoin. Proposition 4.3 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 4.8: Let $*(X_1, X_2, \ldots, X_n)$ be a join dependency on scheme R with zero order attributes $E_R = X_1 \cup X_2 \cup \cdots \cup X_n$. The *decomposition union* (or Δ*-union*) of two 1NF relations r_1 and r_2 on R is

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

where $\bowtie$ is the standard natural join.

Proposition 4.4: 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 $\neg$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. □

4.2.2 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.

Definition 4.9: 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:

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

Proposition 4.5: 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 4.6: 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, $\forall t' \in T : t' \in \mu^*(r) \cap \mu^*(q)$. □

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 $r_1, r_2 \in Rel$.

4.2.3 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.

Definition 4.10: 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:

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

Proposition 4.7: Extended difference is *faithful* to standard difference.
Proof: Similar to proofs for union and intersection. □

Proposition 4.8: Extended difference is *not* a *precise* generalization of standard difference with respect to unnesting.

r_1

A	B*	C*
	B	C
a	b	c
		c′

r_2

A	B*	C*
	B	C
a	b′	c

$r_1 -^e r_2$

A	B*	C*
	B	C
a	b	c′

$\mu^*(r_1 -^e r_2)$

A	B	C
a	b	c′

$\mu^*(r_1)$

A	B	C
a	b	c
a	b	c′

$\mu^*(r_2)$

A	B	C
a	b′	c

$\mu^*(r_1) - \mu^*(r_2)$

A	B	C
a	b	c
a	b	c′

Figure 4-5. Counterexample to preciseness of $-^e$.

Proof: Figure 4-5 shows two ¬1NF relations r_1 and r_2 where $\mu^*(r_1 -^e r_2) \neq \mu^*(r_1) - \mu^*(r_2)$. □

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 4-5, 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 ¬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 4.11: Let $*(X_1, X_2, \ldots, X_n)$ be a join dependency on scheme R with zero order attributes $E_R = X_1 \cup X_2 \cup \cdots \cup X_n$. The *decomposition difference* or *Δ-difference*, of two 1NF relations r_1 and r_2 on R is

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

where $\bowtie$ is the natural join.

Proposition 4.9: 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 ¬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. $\square$

4.2.4 Cartesian Product and Select

The standard product and select operators can be used on ¬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 renested according to the user's desires.

More sophisticated predicates for *select* could be defined using set comparison operators (see [AB, Schl]), 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 [Jae2] 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 paper.

4.2.5 Extended Natural Join

Join operations are difficult to define in the ¬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 4.12: 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 4.10: 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. $\square$

Proposition 4.11: 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.

$\subseteq$ 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 intersection, 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 $\mu^*(r) \bowtie \mu^*(q)$. $\square$

4.2.6 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 4.13: The *extended projection* of relation r on attributes X is

$$\pi^e_X(r) = \bigcup^e_{t \in \pi_X(r)} (t)$$

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

Proposition 4.12: 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. $\square$

Proposition 4.13: Extended projection is *precise* generalization of standard projection with respect to unnesting.

Proof: We need to show that $\pi_{X'}(\mu^*(r)) = \mu^*(\pi^e_X(r))$, 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 $\pi^e_X(r)$, 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 $\mu^*(\pi^e_X(r))$ such that all tuples in T unnested from a single tuple t in $\pi^e_X(r)$. 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 [RKS], 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. □

5. Introducing Null Values into the ¬1NF Relational Model

Previous research on nulls in ¬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 the previous section *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 [AB], 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 ¬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 with the relation containing only the tuple $\langle \mathbf{ni} \rangle$. 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.

5.1 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 ¬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 5.1: 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

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 5-1. A sample relation on the Emp scheme.

(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.

Example 5.1: Recall the Emp scheme and sample relation (shown again in Figure 5-1) introduced in the previous section. 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 $\langle$ Jones, $\{\}$, $\{\}$ $\rangle$, or, equivalently, $\langle$ Jones, $\{\langle$ **ni**, **ni** $\rangle\}$, $\{\langle$ **ni**, $\{\langle$ **ni**, **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 $\langle$ Jones, $\{$**dne**, **dne**$\}$, $\{\langle$ **unk**, $\{\langle$ 1981, **unk** $\rangle\}\}$ $\rangle$. $\square$

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

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

According to definition 5.1, this tuple is less informative than the one in Figure 5-1. 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 5.2: Let $t_1 = \langle$ Smith, $\{\langle$ Sam $\rangle, \langle$ Sue $\rangle\}$ $\rangle$ and $t_2 = \langle$ Smith, $\{\langle$ Sue $\rangle, \langle$ Bill $\rangle\}$ $\rangle$ be tuples from a projected employee relation. We have that $t_1 \not\geq t_2$ and $t_2 \not\geq t_1$, but under PNF t_1 and t_2 would be combined into $t_3 = \langle$ Smith, $\{\langle$ Sam $\rangle, \langle$ Sue $\rangle, \langle$ Bill $\rangle\}$ $\rangle$. $\square$

The definitions of *x-element* ($\widetilde{\in}$), and *tuple set reduction* ($\widehat{\{set\ of\ tuples\}}$), from section 3, 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 5.2: 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 **ni** 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.

5.2 Operators for ¬1NF Relations with Nulls

Since the mapping between 1NF and ¬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 ¬1NF operators defined in the previous section 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.

5.2.1 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 5.3: Consider the 1NF relation of Figure 5-2a. 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 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 5-2b 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 5.3. 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 ⟨ **ni**, {⟨ Science1 ⟩} ⟩ and ⟨ **ni**, {⟨ Science2 ⟩} ⟩ tuples with the single tuple, ⟨ Jones, {⟨ Science1 ⟩, ⟨ Science2 ⟩} ⟩. 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:

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 5-2. Example of *nest* with null values.

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

Proposition 5.1: 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 $\hat{p} \in POSS^*(p)$. Tuples in $\hat{p}$ may be a combination of tuples in p only if $POSS^*$ assigned the same value to nulls in otherwise equal partition keys of p. By making this same assignment directly to r, and then nesting, we can generate $\hat{p}$. 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 $POSS^*$. □

5.2.2 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 5.4: Recalling the database scheme of the previous example, consider a relation r with two tuples $t_1 = \langle$ Jones, $\{\langle$ Math $\rangle$, $\langle$ Science $\rangle\}$ $\rangle$ and $t_2 = \langle$ **ni**, $\{\langle$ Math $\rangle$, $\langle$ English $\rangle\}$ $\rangle$. If we unnest r, then the resulting $\langle$**ni**, Math$\rangle$ tuple is less informative than the $\langle$Jones, 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 [Sci]. 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 5.4, every relation in $\mu_{course*}(POSS^*(r))$ must contain $\langle x$, Math $\rangle$ and $\langle x$, English $\rangle$ for some value x. However, there are relations in $POSS^*(\mu'_{course*}(r))$ 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 5.2: 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 $\widehat{p} \in POSS^*(p)$. If we make the same assignment to the nulls in p as in the nested relation r then $\widehat{p} \in \mu_B(POSS^*(r))$. 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 $\widehat{p} \in \mu_B(POSS^*(r))$. Then there must be $\widehat{r} \in POSS^*(r)$ such that $\widehat{p} = \mu_B(\widehat{r})$. Let t_p be a tuple in p. Now, t_p unnested from some tuple t_r in r, which has some PNF possibility $t_{\widehat{r}} \in \widehat{r}$ such that $t_{\widehat{r}} \geq t_r$. Let $t_{\widehat{p}} = \mu_B(t_{\widehat{r}})$. Then, we have $t_{\widehat{p}} \geq t_p$. We conclude that $\widehat{p} \geq p$ and so $\widehat{p} \in POSS^*(p)$.

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

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

Corrolary 5.1: Null-unnest* is a precise generalization of standard unnest* with respect to PNF possibility function $POSS^*$.

Proof: Apply the same argument as for Proposition 5.2, only use complete unnesting instead of single unnesting. $\square$

5.3 Null-extended Operators

Let $Rel\uparrow^*$ represent the set of all relations which are not in 1NF or 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 5.3: 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 5.4: 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 5.1: Definitions 5.3 and 5.4 are equivalent.

Proof: By Corrolary 5.1, we know that null-unnest* is a precise generalization of standard unnest* for $POSS^*$. Thus, the definitions are equivalent. □

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 previous sections, proofs of faithfulness are straightforward and so we shall omit them here.

5.3.1 Null-extended union

Our definition of null-extended union can be revised to accommodate nulls by adding tuple set reduction as follows.

Definition 5.5: 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:

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

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

Proposition 5.3: 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 4.4, 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 t 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, $POSS^*(p) \supseteq POSS^*(r) \cup^e POSS^*(q)$.

$\subseteq$ 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, $\hat{p} \in POSS^*(r) \cup^e POSS^*(q)$, and so $POSS^*(p) \subseteq POSS^*(r) \cup^e POSS^*(q)$.

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

5.3.2 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 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 and Z range over the higher order attributes in E_R. The *null-extended difference* of r_1 and r_2 is:

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

Proposition 5.4: Null-extended difference is a *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 that there does not exist p such that $\mu^*(POSS^*(r -^{e'} q)) \supsetneq \mu^*(POSS^*(p)) \supseteq \mu^*(POSS^*(r)) -^{\Delta} \mu^*(POSS^*(q))$. By Proposition 4.9, 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 there does not exist p such that $POSS^*(r -^{e'} q) \supsetneq POSS^*(p) \supseteq POSS^*(r) -^{e} POSS^*(q))$. Suppose there is some p. If $POSS^*(r -^{e'} q) \supsetneq POSS^*(p)$, 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 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 t' be the tuple and z be the relation (either r or a nested relation in r) where the matching does not occur, and w be the corresponding 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-difference is a restricted generalization of standard difference for $POSS^*$. □

5.3.3 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 $POSS^*$. As in the previous two sections we will use the standard cartesian product operator. For select we will use *null-select* as defined in section 3.

5.3.4 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 5.7: 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 5.5: 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 4.11 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 $\widehat{p} \in POSS^*(r) \bowtie^{e} POSS^*(q)$. Also, let C be the common zero order attributes of r and q. Then, there must be $\widehat{r} \in POSS^*(r)$ and $\widehat{q} \in POSS^*(q)$ such that $\widehat{p} = \widehat{r} \bowtie^{e} \widehat{q}$. 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 $t_{\widehat{r}} \in \widehat{r}$ and $t_{\widehat{q}} \in \widehat{q}$ that agree on C and will participate in the join giving $t_{\widehat{p}}$. Now, the common higher order attributes X of $t_{\widehat{r}}$ and $t_{\widehat{q}}$ 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, $t_{\widehat{p}} \geq t_p$, $\widehat{p} \geq p$, and so $\widehat{p} \in POSS^*(p)$.

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, there are possibilities which do not have those different actual values. 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 $POSS^*$.

5.3.5 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 5.8: The *null-extended projection* of relation r on attributes X is

$$\pi'^{e}_{X}(r) = \widehat{\{t \mid t \in \pi^{e}_{X}(r)\}} = \bigcup_{t \in r[X]}^{e'} (t)$$

Proposition 5.6: 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 3.3). □

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 6-1. Relation satisfying $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow C$, but not $A \rightarrow\rightarrow C$ when **dne** nulls are not equated.

6. Dependencies in a ¬1NF Database with Null Values

A key assumption made in this paper 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 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.

6.1 Non-existent Nulls

In [Lie2], 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 $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow C$ hold, but $A \rightarrow\rightarrow C$ does not hold (Figure 6-1).

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 examing 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 $\langle a_3, b, \mathbf{dne}, d_3\rangle$ 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 $\mathcal{O}$. For each $O \in \mathcal{O}$, in $\pi_O(r)$ there does not exist two tuples t_1 and t_2 where $t_1[A] = \mathbf{dne}$, $t_1[A] \neq t_2[A]$, and $t_1[O - A] = t_2[O - A]$, for any $A \in O$.

6.2 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 [Lie1, Lie2] 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 6-1, 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.

6.3 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 [Lie2], about the lack of transitivity in this model. Based on the lattice developed in section 2, 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**.

6.4 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 $\langle a, \mathbf{ni}, c\rangle$ 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, $B \rightarrow\!\!\!\rightarrow A$, 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.

7. Conclusion

The model of incomplete information presented in this paper is based on the concept of *more informative* tuples and relations. Using a partial order in which the *no-information* null is less informative than both the *unknown* and *non-existent* nulls allows systems to be designed with either the no-information null alone or with a combination of nulls. If one wants to avoid any computational problems with unknown nulls, they can be deleted from the model. However, the framework is there if applications arise in which the no-information interpretation is not adequate.

It was shown how the theory of nulls can be used in a $\neg$1NF database with a straightforward extension. We discovered that our $\neg$1NF extended operators have a pleasing mapping to their 1NF counterparts, based on the concept of *partitioned normal form.* Furthermore, null values do not affect the operation of the important *nest* and *unnest* operators. Finally, we showed how existing theories on the axiomatization of functional and multivalued dependencies in the presence of nulls are flawed, and, in fact, the traditional axiomatization is valid.

Further work is needed in the area of relational operators for $\neg$1NF relations. We especially need more sophisticated select and project operators which can work on nested relations. The lack of a satisfactory generalization for natural join suggests that more work is necessary before a solution is reached. Of special interest is a join which will work in a database in nested normal form. This topic and other language issues will be the topic of forthcoming papers.

8. Bibliography

[AB] 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).

[AM] Atzeni, P. and N. Morfuni, "Functional Dependencies in Relations with Null Values," *Information Processing Letters 18*, 4 (May 1984), 233–238.

[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.

[Cod] Codd, E., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems 4*, 4 (December 1979), 397–434.

[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.

[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).

[FV2] Fischer, P. and D. Van Gucht, "Weak Multivalued Dependencies," *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Waterloo (April 1984), 266–274.

[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.

[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.

[Jae1] Jaeschke, G., "Nonrecursive Algebra for Relations with Relation Valued Attributes," TR 84.12.001, Heidelberg Scientific Center, IBM Germany (December 1984).

[Jae2] Jaeschke, G., "Recursive Algebra for Relations with Relation Valued Attributes," TR 84.01.003, Heidelberg Scientific Center, IBM Germany (January 1984).

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

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

[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.

[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.

[LP] LaCroix, M. and A. Pirotte, "Generalized Joins," *ACM SIGMOD Record 8*, 3 (September 1976), 14–15.

[Lie1] 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.

[Lie2] Lien, Y., "On the Equivalence of Database Models," *Journal of the ACM 29*, 2 (April 1982), 333–362.

[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.

[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.

[OO] 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.

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

[PHH] Pistor, P., B. Hansen and M. Hansen, "Eine sequelarige Sprachschnittstelle für das NF2-Modell." In *Sprachen für Datenbanken*, J. Schmidt, Ed., Informatik Fachberichte Nr. 72, Springer-Verlag, Berlin, 1983.

[RKB] Roth, M., H. Korth and D. Batory, "SQL/NF: A Query Language for ¬1NF Relational Databases," TR-85-19, Department of Computer Science, University of Texas at Austin (September 1985).

[RKS] Roth, M., H. Korth and A. Silberschatz, "Theory of Non-First-Normal-Form Relational Databases," TR-84-36, Department of Computer Science, University of Texas at Austin (December 1984).

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

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

[SS1] 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).

[SS2] 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.

[Sci] Sciore, E., "Null Values, Updates, and Normalization in Relational Databases," Technical Report, Department of Electrical Engineering and Computer Science, Princeton University (December 1979).

[TF] Thomas, S. and P. Fischer, "Nested Relational Structures." In *The Theory of Databases*, P. Kanellakis, Ed., JAI Press, to appear, 1985.

[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.

[Won] Wong, E., "A Statistical Approach to Incomplete Information in Database Systems," *ACM Transactions on Database Systems 7*, 3 (September 1982), 470–488.

[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.