Acta Informatica 26, 615–642 (1989)

Null Values in Nested Relational Databases

Mark A. Roth*, Henry F. Korth**, and Abraham Silberschatz
Department of Computer Science, University of Texas at Austin, Austin, TX 78712, USA

Summary. The desire to extend the applicability of the relational model beyond traditional data-processing applications has stimulated interest in nested or 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 nested relational model using an open world assumption. We extend the traditional theory and study the properties of extended operators for nested 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, nested or non-first normal form relations were proposed in which the attributes of a relation can take on values which are sets or even relations themselves. This new assumption created a need to reexamine the fundamentals of relational database theory, and opened the door for the introduction of operators which take advantage of the *nested* structure of nested relations.

* Currently with the Air Force Institute of Technology, AFIT/ENG, Wright-Patterson AFB, OH 45433, USA

** Research partially supported by an IBM Faculty Development A ward and NSF grant DCR-8507224

Offprint requests to: M.A. Roth

This paper is concerned with the representation of null values in nested relational databases and the definition of algebraic operators on nested relations containing nulls. Null values have traditionally represented the nonexistence of a value or the fact that a value is unknown. In 1NF relational databases it is not always necessary to represent missing data by means of null values. To illustrate this, consider a database consisting of two relations r_1 (employee, skill) and r_2 (employee, child). Skills can be stored for an employee, Jones with no children by adding tuples to r_1. No tuple appears in r_2 for Jones. However, in a nested relational database, we would likely have a single relation r whose attributes are *employee*, *skill*, and a set-valued *children* attribute. Jones would have the empty set of children. The empty set is, in a sense, a null value, since unnesting the relation r forces us to introduce nulls into the resulting 1NF relation.

Thus, the need for nulls is even more critical in a nested database than a 1NF database. Since we have the ability to represent multiple relationships in a single nested relation without the problems of redundancy that doing so in a 1NF relation would entail, we must also deal with the fact that one or more of those relationships may be unknown or non-existent at some time.

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.

The remainder of this paper is organized as follows. In Sect. 2, we define the nested relational 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 nested relations. In Sect. 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. In Sect. 4, we extend the null value theory presented in Sect. 3 to nested relations. In Sect. 5 we introduce the extension of algebraic operators to relations with null values. We define the concepts used to measure the "goodness" of these extensions. Section 6 provides the actual extension of the algebra to nulls along with proofs of properties of our extensions. Finally, in Sect. 7, we discuss dependency theory, shedding some new light on the problem of nulls when dealing with functional and multivalued dependencies, and their axiomatization.

2. The Nested Relational Model

In this section, we briefly review concepts from the relational and nested relational models. 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 [22] and led to the concept of nesting. This was later studied by Jaeschke and Schek [12] for one level nesting over single attributes and by Thomas and Fischer [38] in a more general setting. Utilizing nested relations for structuring database outputs was discussed by Kambayashi et al. [13], while Fischer

and Van Gucht [6, 7] looked at dependencies which characterize nested relations.

Özsoyoğlu and Özsoyoğlu [26] consider operations similar to that of [12], and extend the basic algebra for relations by aggregate operators. Our previous work [32] defines a relational calculus and relational algebra for nested relations and proves their equivalence. We also introduced *partitioned normal form* for nested relations (described later) which is equivalent to *scheme trees* of [27] and *formats* of [1]. Abiteboul and Bidoit [1] also define some extended operators which are refined in [32], where it was also proved that the set of relations in partitioned normal form are closed under the extended operators. Others [10, 11, 28, 31, 33–35] have been developing languages and implementations for nested relational databases.

2.1. *Nested Relational Schemes*

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* $\mathscr{R}$ 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 $A_i \in U$, $1 \leqq i \leqq n$. 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 $\cup, \cap, -, \times, \bowtie, \pi$, 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. We assume familiarity with 1NF relations and the relational algebra on these relations. We also assume familiarity with the definitions of data dependencies (functional, multivalued, and join). For these definitions, see a database text such as [15]. Fixing a particular scheme, the set of all relations on that scheme that satisfies a set of dependencies D is denoted $SAT(D)$.

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}, $1 \leqq i \leqq n$, 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.

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

Fig. 1. A sample relation on the Emp scheme

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. Nested 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. A nested relation is represented simply as a higher order attribute on the right hand side of a rule.

Example 1. Consider an 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).

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 Fig. 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.

2.2. *Partitioned Normal Form*

In the relation of Fig. 1, we do 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 leads us to restrict the set of nested relations to those that are in *partitioned normal form* (PNF).

A relation r is in PNF if the zero order attributes in the relation r form a key for r, and each nested relation of r is also in PNF. The employee relation in Fig. 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 1 [32]. 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 $X_1, X_2, \ldots, X_l$. Relation r is in *partitioned normal form* (PNF) if and only if the following two conditions hold:

1. $A_1 A_2 \ldots A_k \rightarrow E_R$.
2. For all $t \in r$ and for all X_i: $1 \leqq i \leqq l$: $\mathscr{R}_{ti}$ is in PNF, where $\mathscr{R}_{ti}$ is the nested relation $t[X_i]$ on scheme X_i.

2.3. *Nest and Unnest Operators*

We consider now operators for nested relations without null values. The traditional relational algebra operators can be used with nested relations. We add *unnest* and *nest* operators to allow restructuring of nested relations. We state the formal definitions of *nest* and *unnest* from [32, 38].

Definition 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. $R' = (C, (B_1, B_2, \ldots, B_m)) = (\mathrm{C}, \mathrm{B})$ 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 $t[C] = u[C] \wedge$
$t[B] = \{v'[B_1 B_2 \cdots B_m] \mid v \in r \wedge v[C] = t[C]\}\}$.

Definition 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 $t[C] = u[C]$
$\wedge\, t[B_1 B_2 \ldots, B_m] \in u[B]\}$.

We will use μ^* or unnest* to indicate the complete unnesting of a relation into a 1NF relation.

There is not much correspondence between the way most of the relational algebra operators work on 1NF relations and their counterparts work on nested relations.

Example 2. Consider nested relations r_1 and r_2 of Fig. 2 and their 1NF counterparts, s_1 and s_2. Note, however, that $r_1 \cap r_2$ is not the nested 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. □

r_1		r_2		$r_1 \cap r_2$	
A	B* B	A	B* B	A	B* B
a	b	a	b	a	b
	b′		b′		b′
a′	b	a′	b		
	b′		b″		

s_1		s_2		$s_1 \cap s_2$	
A	B	A	B	A	B
a	b	a	b	a	b
a	b′	a	b′	a	b′
a′	b	a′	b	a′	b
a′	b′	a′	b″		

Fig. 2. Intersection applied to nested and 1NF relations

We believe that each 1NF operator should have a *reasonable* nested counterpart. Intuitively, a nested 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 nested relations. We take this point of view since the information content of a nested relational database is the same as in the corresponding 1NF relational database. We thus desire the ability both to take advantage of the nested structure and easily to simulate the effect of the 1NF operators on 1NF relations. In [29, 32], we define in terms of the basic operators extended operators that operate on nested relations and have this "reasonableness" property. In that paper, we show by example that the extended operators preserve the PNF property and, thus, are more appealing intuitively in most cases than direct application of the traditional algebra operators.

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 [42, 43]. 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 [3, 4, 9, 20], and *nonexistent* by [16, 17, 37, 43]. In [40]

$b \backslash a$	d_1	d_2	...	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**
⋮	**unk**	**unk**		**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**

Fig. 3. Definition of glb function

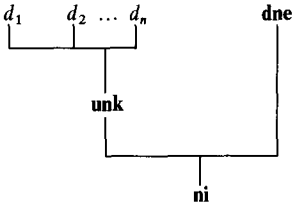

Fig. 4. Information lattice

a combination of the two is proposed in which nonexistence is considered an inconsistent state of data. Finally, Zaniolo [42] 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 nested relations.

Other proposals for nulls are rather sophisticated, involving partial specification [18, 19, 23–25], probability distributions [41], and conditional tuples [14], but it could be argued "that the complexity of their management is not justified by their richer semantics" [2, p. 233].

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

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 Fig. 4).

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, **dne** = **dne**, but **ni** ≠ **ni** and **unk** ≠ **unk**.

Our decision to define equality in this manner is consistent with our open world assumption. Assuming **ni** ≠ **ni** and **unk** ≠ **unk** does not preclude the potential existence of values such that equality holds. Rather our assumption is consistent with the fact that we cannot prove equality based on the information given in the database.

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

Definition 5. An X-value s is said to be *more informative* than a Y-value t, written $s \geqq t$, 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 $s \geqq t$ 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 $s \geqq t$. 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 $t \geqq s$ and $s \geqq t$, 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. Let

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

denote E_{R_2}-values, and let

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

denote E_{R_1}-values. Then, t_1 is more informative than t_2, t_3, and t_4. Furthermore, $t_4 \geqq t_2$, $t_4 \geqq t_3$, 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 6. 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 4. Using the tuples defined in Example 3 we find that

$$t_1 \wedge t_3 = t_2$$
$$t_1 \wedge t_4 = \langle \text{Smith}, \textbf{ni}, \text{typing} \rangle. \quad \square$$

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

Definition 7. 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 $s \geqq t$.

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

$$\hat{\{}t_1, t_2, \ldots, t_n\hat{\}}.$$

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

Definition 8. A relation r_1 is *more informative than*, or *subsumes*, a relation r_2, written $r_1 \geqq r_2$, when for each tuple $t_2 \in r_2$ there is a tuple $t_1 \in r_1$ with $t_1 \geqq t_2$.

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

Definition 9. The relations r_1 and r_2 are *information-wise equivalent*, written $r_1 \cong r_2$, when $r_1 \geqq r_2$ and $r_2 \geqq r_1$.

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

Definition 10. A relation r constitutes a *minimal representation* for a relation q when $r \subseteq q$, $r \cong q$, and $\nexists p \subset r$ such that $p \cong q$.

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

4. Null Values in Nested Relations

We extend the definitions of the previous section to handle nested relations. The key issue is the interpretation of empty sets. Schek states, "In the general case unnest on empty relations will produce undefined attribute values" [33, page 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 [1], 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.

Several researchers have assigned the *non-existent* interpretation to empty set. One of Makinouchi's properties of "not-necessarily-normalized" relations

[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

is that "A null set ($\emptyset$) may be in the domain of a relation column. $\emptyset$ means exactly non-existence" [22, p. 448]. In deriving an extended set-containment operation for 1NF relations with non-existent nulls, Zaniolo [43] discusses the nested 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. Scholl [36] assigns the non-existent interpretation to empty set in his model utilizing a closed-world assumption. This is appropriate when utilizing nested relations to store data from a 1NF database. Then the empty set appears only when dangling tuples are present.

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 $\langle$**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. Our work inspired Güting et al. [8], who have taken a similar approach but rename the null values so that empty set corresponds to the atomic **dne** null and a new set-valued null, **niseq**, corresponds to the atomic **ni** null for nested relations. We feel our approach is simpler and leads to more straightforward extensions of null value concepts for the nested relational model.

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 nested 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 11. Let t_1 be a tuple on zero order attributes X_1 and higher order attributes Y_1, and let t_2 be a tuple on zero order attributes X_2 and higher order attributes Y_2. The tuple t_1 is said to be *more informative* than the tuple t_2 when:

1. for each $B \in X_2$, if $t_2[B]$ is not **ni** then $B \in X_1$,
2. for each $C \in Y_2$, if $t_2[C]$ contains a tuple that is not null then $C \in Y_1$,
3. for each $A \in X_1 \cap X_2$, $glb(t_1[A], t_2[A]) = t_2[A]$, and
4. 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. Recall the Emp scheme and sample relation introduced in the previous section (see Fig. 1). 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 ⟨Jones, { }, { }⟩, or, equivalently, ⟨Jones, {⟨**ni**, **ni**⟩}, {⟨**ni**, {⟨**ni**, **ni**⟩}⟩}⟩. 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 ⟨Jones, {⟨**dne**, **dne**⟩}, {⟨**unk**, {⟨1981, **unk**⟩}⟩}⟩. □

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

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

According to Definition 11, this tuple is less informative than the one in Fig. 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 nested 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 6. Let $t_1 = \langle \text{Smith}, \{\langle \text{Sam}\rangle, \langle \text{Sue}\rangle\}$ and $t_2 = \langle \text{Smith}, \{\langle \text{Sue}\rangle, \langle \text{Bill}\rangle\}\rangle$ be tuples from a projected employee relation. We have that $t_1 \not\geq t_2$ and $t_2 \not\geq t_1$, but under PNF t_1 and t_2 would be combined into $t_3 = \langle \text{Smith}, \{\langle \text{Sam}\rangle, \langle \text{Sue}\rangle, \langle \text{Bill}\rangle\}\rangle$. □

The definitions of *x-element* ($\tilde{\in}$), and *tuple set reduction* ($\hat{\{}$*set of tuples*$\hat{\}}$), from Sect. 3, carry over to nested relations in a straightforward manner. However, the *meet* of two 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 12. Let U be the attributes on which two tuples t_1 and t_2 are defined, where t_1 and t_2 have been extended to U with the addition of **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 $X \in U$, $t[X] = \{s \wedge u | s \in t_1[X] \text{ and } u \in t_2[X]\}$.

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 nested version of subsumption.

5. Extended Operators

In this section, we extend the relational algebra in several steps. First, we consider extending the 1NF algebra to include nulls, then an extension to the nested relational algebra, and finally the extension of the nested relational algebra

to include nulls. As before, we assume that all relations are in PNF. Our definition of PNF relies on the definition of functional dependency in which we test equality of attribute values. As discussed in Sect. 3, **ni** ≠ **ni**, **unk** ≠ **unk**, and **dne** = **dne**. 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. Here we are concerned with only the single application of operators rather than composition of operators. See [24] for a treatment of issues with null values in 1NF relational expressions. Some of the following presentation is based on [21, Sect. 12.4].

5.1. *Classes of Relations*

In order to define our extensions to the relational algebra, we define classes of relations corresponding to 1NF relations and nested relations, with and without nulls:

- *Rel*: the set of all 1NF relations having no nulls
- *Rel*↑: the set of all 1NF having at least one null value
- *Rel**: the set of all nested relations having at least one higher order attribute
- *Rel*↑*: the set of all nested relations having at least one higher order attribute or at least one null value.

We denote the restriction of *Rel*, *Rel*↑, *Rel**, and *Rel*↑* to scheme R by $Rel(R)$, $Rel{\uparrow}(R)$, $Rel^*(R)$, and $Rel{\uparrow}^*(R)$ respectively. Observe that $Rel^* \cup Rel{\uparrow} = Rel{\uparrow}^*$ and $Rel \cap Rel{\uparrow}^* = \emptyset$.

5.2. *Possibility Functions*

We relate a relation containing null values with the set of null-free relations that subsume it. This relationship is defined by a *possibility function*. We shall consider two such functions:

- *POSS*, relating *Rel* and *Rel*↑
- *POSS**, relating *Rel* and *Rel*↑*.

A relation r in $Rel{\uparrow}(R)$ represents a set of relations from $Rel(R)$ that subsume r. Each such relation in $Rel(R)$ is called a *possibility* for r. The set of possibilities for r is denoted by $POSS(r)$, which is defined as:

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

The above definition could be applied to nested relations as well. However, this could result in a PNF relation having non-PNF possibilities. To illustrate this, consider a relation containing the two tuples $\langle \mathbf{ni}, \{\langle a \rangle\} \rangle$ and $\langle \mathbf{ni}, \{\langle b \rangle\} \rangle$. If we allow both **ni** nulls to be replaced by the same value x, the result is a non-PNF relation containing $\langle x, \{\langle a \rangle\} \rangle$ and $\langle x, \{\langle b \rangle\} \rangle$. We choose instead to consider only PNF possibilities and would thus consider $\langle x, \{\langle a \rangle, \langle b \rangle\} \rangle$

in this case. Formally, the set of *PNF possibilities* for relation r on scheme R is denoted $POSS^*(r)$, and is defined as:

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

5.3. *Faithfulness and Precision of Generalizations*

In this section we present the criteria that we use to establish the correctness of our extended operators for nested relations with null values. We start by extending the definition of relational operators to map sets of relations to other sets of relations.

Definition 13. For sets P_1 and P_2 of relations and relational operator γ,

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

and

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

Our criteria for correctness of an extended operator is that it be *faithful* and *precise.*

Definition 14. Let P and P' be classes of relations and γ and γ' operators on P and $P \cup P'$ respectively.

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 $r \in P$ for which $\gamma(r)$ is defined.

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

Definition 15. Let P and P' be classes of relations and γ and γ' operators on P and P' respectively. Let α be an operator on $P \cup P'$. We say γ' is a *precise* generalization of γ relative to α if one of the following two conditions holds:

1. When γ and γ' are unary operators, $\alpha(\gamma'(r)) = \gamma(\alpha(r))$ for every $r \in P'$.

2. When γ and γ' are binary operators, $\alpha(r \gamma' q) = \alpha(r)\ \gamma\, \alpha(q)$ for every $r, q \in P'$.

Our generalization of relational operators to sets of relations allows α to map relations to sets of relations. We shall use this subsequently when we consider α to be a possibility function.

We shall see that for some choices of α, not all relational operators have a precise generalization relative to α. In these cases, we consider the weaker notion of an *adequate* and *restricted* generalization which captures $\gamma(\alpha(r))$ or $\alpha(r)\ \gamma\, \alpha(\gamma)$ and as little extra as is possible.

Definition 16. Let P and P' be classes of relations, and γ and γ' be operators on P and P' respectively. Let α be an operator on $P \cup P'$. We say that operator

γ' is an *adequate* generalization for γ with respect to α if one of the following two conditions holds:

1. When γ and γ' are unary operators, $\alpha(\gamma'(r)) \supseteq \gamma(\alpha(r))$ for every $r \in P'$.
2. When γ and γ' are binary operators, $\alpha(r \gamma' q) \supseteq \alpha(r)\ \gamma\alpha(q)$ for every $r, q \in P'$.

Furthermore, we say that operator γ' is a *restricted* generalization for γ with respect to α if one of the following two conditions holds:

1. When γ and γ' are unary operators, for every $r \in P'$, there is no p in P' such that $\alpha(\gamma'(r)) \supsetneq \alpha(p) \supseteq \gamma(\alpha(r))$.
2. When γ and γ' are binary operators, for every $r, q \in P'$, there is no p in P' such that $\alpha(r \gamma' q) \supsetneq \alpha(p) \supseteq \alpha(r)\ \gamma\alpha(q)$.

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

Generalizations of the standard operators relative to *POSS* appear in [42, 43]. These generalizations are faithful, and at least adequate and restricted, if not precise.

6. Nested Operators

We now define nested operators which are both faithful and precise and, moreover, they also have some intuition behind them. In [32], 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.

In order to take the *extended union* of two relations r_1 and r_2 we require that they be defined over equal relation schemes, say R. The scheme of the resultant structure is also R. We define *extended union*, denoted by $\cup^e$, at the instance level as follows.

Definition 17. Let r_1 and r_2 be relations on scheme R. Let X range over the zero order attributes in E_R, and let 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			r_2			$r_1 \cup^e r_2$			$\mu^*(r_1 \cup^e r_2)$		
A	B*	C*	A	B*	C*	A	B*	C*	A	B	C
	B	C		B	C		B	C			
a	b	c	a	b′	c	a	b	c	a	b	c′
		c′					n′	c′	a	b	c′
									a	b′	c
									a	b′	c′

$\mu^*(r_1)$			$\mu^*(r_2)$			$\mu^*(r_1) \cup \mu^*(r_2)$		
A	B	C	A	B	C	A	B	C
a	b	c	a	b′	c	a	b	c
a	b	c′				a	b	c′
						a	b′	c

Fig. 5. Counterexample to preciseness of $\cup^e$

Extended union is *not* a precise generalization of standard union with respect to unnesting. Figure 5 shows two nested 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.

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 18. 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*), denoted by $\cup^\Delta$, 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.

Extended difference, denoted by $-^e$ 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 19. Let r_1 and r_2 be relations on scheme R. Let X range over the zero order attributes in E_R and let Y and Z range over the higher order attributes in E_R. The *extended difference* of r_1 and r_2 is:

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

The *extended difference* operator has semantic complications similar to extended union. Extended difference is *not* a *precise* generalization of standard difference with respect to unnesting. This leads us to define the decomposition difference.

Definition 20. 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*, denoted by $-^{\Delta}$, 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.

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 21. 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, denoted by $r_1 \bowtie^e r_2$ which produces a relation r on scheme R where:

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

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 22. The *extended projection* of relation r on attributes X, denoted by π^e is:

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

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

The following propositions summarize the faithfulness and preciseness results for the extended operators. Detailed proofs for all operators can be found in [29].

Proposition 1. *Extended union, intersection, difference, natural join, and projection are faithful to standard union, intersection, difference, natural join, and projection, respectively.*

Proposition 2. *Extended union and extended difference are precise generalizations of Δ-union and Δ-difference with respect to unnesting.*

Proposition 3. *Extended intersection is a precise generalization of standard intersection with respect to unnesting.*

Proposition 4. *Extended natural join is precise generalization of standard natural join with respect to unnesting.*

Proposition 5. *Extended projection is a precise generalizatin of standard projection with respect to unnesting.*

r		$\nu'_{\text{course}*=(\text{course})}(r)$	
Teacher	Course	Teacher	Course* Course
Smith	Math 1	Smith	Math 1
Smith	Math 2		Math 2
dne	Math 5	**dne**	Math 5
dne	Math 6		Math 6
ni	Science 1	**ni**	Science 1
ni	Science 2	**ni**	Science 2
(a)		(b)	

Fig. 6. Example of *nest* with null values

6.1. Null Nest and Unnest

In this section we consider the semantics of nest and unnest in the presence of nulls.

6.1.1. 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* (ν') 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. This is consistent with our definition in Sect. 3 of equality applied to nulls.

Example 7. Consider the 1NF relation of Fig. 6a. 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 "Math 1" and "Math 2" together in a nested relation. The same applies to the two tuples with **dne** nulls. These two tuples indicate that "Math 5" and "Math 6" are courses that exist, but there are not 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 "Math 5" 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 "Math 2".

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 6b shows the nested relation. □

Consider the nested relation of Example 7. Using *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**, {⟨Science 1⟩}⟩ and ⟨**ni**, {⟨Science 2⟩}⟩ tuples with the single tuple, ⟨Jones, {⟨Science 1⟩, ⟨Science 2⟩}⟩. The resulting relation is in PNF. Therefore, we will use *POSS**, rather than *POSS* as our possibility function.

Proposition 6. *Null-nest* (ν') *is a precise generalization of standard nest* (ν) *with respect to POSS*.*

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

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

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

We conclude that ν' is a precise generalization of ν for *POSS**. □

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

The problem with arbitrary nested 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 [37]. 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. Although we do not deal explicitly with marked nulls in this paper, our results extend naturally to a model that includes them.

Another reason for our treatment of **ni** and **unk** is so that null-unnest is a precise generalization of the standard operator. In Example 8, every relation in $\mu_{course*}(POSS*(r))$ must contain $\langle x, \text{Math}\rangle$ and $\langle x, \text{English}\rangle$ for some value x. However, there are relations in $POSS*(\mu'_{course*}(r))$ 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 (μ') is unchanged from the standard unnest definition. Furthermore, we can prove that null-unnest is a precise generalization.

Proposition 7. *Null-unnest* (μ') *is a precise generalization of standard unnest* (μ) *with respect to POSS*.*

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

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

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

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

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

Corollary 1. *Null-unnest** ($\mu'*$) *is a precise generalization of standard unnest** ($\mu*$) *with respect to POSS*.*

Proof. Apply the same argument as for Proposition 7, only use complete unnesting instead of single unnesting. □

6.2. Null-extended Operators

Let $Rel\uparrow*$ represent the set of all relations which are not in 1NF or which contain a null value. Thus, $Rel* \cup Rel\uparrow = Rel\uparrow*$ and $Rel \cap Rel\uparrow* = \emptyset$. Our goal

is to generalize the nested operators to deal with null values. We have two choices for our definition of a precise generalization for the operators with respect to a composition of unnesting and the PNF possibility function. 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 23. 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 24. 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, $POSS^*(\mu'^*(r\gamma'^* q)) = POSS^*(\mu'^*(r))$ γ $POSS^*(\mu'^*(q))$ for every $r, q \in Rel\uparrow^*$ for which $r\gamma'^* q$ is defined.

Theorem 1. *Definitions 23 and 24 are equivalent.*

Proof. By Corollary 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**. The proofs of faithfulness are straightforward and so we shall omit them in what follows.

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

Definition 25. 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:

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

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

Proposition 8. *Null-extended union is a precise generalization of Δ-union with respect to unnesting and* PNF *possibility function POSS*.*

Proof. We show that $\mu^*(POSS^*(r \cup^{e'} q)) = \mu^*(POSS^*(r)) \cup^{\Delta} \mu^*(POSS^*(q))$. By Proposition 2, 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 $t_{\hat{p}} \geqq t_p$. 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 $\hat{p} \geqq p$, 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 $p \geqq r$, $\hat{p} \geqq r$ 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 $POSS^*$. □

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

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

Proposition 9. *Null-extended difference is an adequate and restricted generalization of Δ-difference with respect to unnesting and possibility function $POSS^*$.*

Proof. We show adequacy and then restrictedness.

adequate: We show $\mu^*(POSS^*(r -^{e'} q)) \supseteq \mu^*(POSS^*(r)) - \mu^*(POSS^*(q))$. By Proposition 2, we know that extended difference is a precise generalization of Δ-difference, and so $\mu^*(POSS^*(r)) -^{\Delta} \mu^*(POSS^*(q)) = \mu^*(POSS^*(r) -^{e} POSS^*(q))$. Thus, we need only show that $POSS^*(r -^{e'} q) \supseteq POSS^*(r) -^{e} POSS^*(q)$. Let $p = r -^{e'} q$, and $\hat{p} \in POSS^*(r) -^{e} POSS^*(q)$. Then, there exists $\hat{r} \in POSS^*(r)$ and $\hat{q} \in POSS^*(q)$, such that $\hat{p} = \hat{r} -^{e} \hat{q}$. Let t_p be a tuple in p. Then, t_p must be in r with, perhaps, some of its needed relations reduced by interaction with a tuple t_q in q. Therefore, there must be tuples $t_{\hat{r}} \in \hat{r}$ and $t_{\hat{q}} \in \hat{q}$ which will also interact in the same way, noting that interaction occurs only when the zero

order attributes have non-null values. Thus there is a tuple $t_{\hat{p}} = t_{\hat{r}} -^{e} t_{\hat{q}}$ in $\hat{p}$, such that $t_{\hat{p}} \geqq t_p$. We conclude that $\hat{p} \geqq p$ and so $\hat{p} \in POSS^*(p)$. Therefore, $POSS^*(p) \supseteq POSS^*(r) -^{e} POSS^*(q)$.

restricted: We show that there does not exist p such that $\mu^*(POSS^*(r -^{e'} q)) \supsetneq \mu^*(POSS^*(p)) \supseteq \mu^*(POSS^*(r)) -^{\Delta} \mu^*(POSS^*(q))$. As in the case for adequate, 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 $POSS^*(r -^{e'} q) \supseteq POSS^*(p)$, 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 z be the relation (either r or a nested relation in r) and t' the tuple in z where the matching does not occur, and w be the corresponding, possibly empty, relation in q. There are two possible reasons for there not being a match: either $t'[X] \in z$ and $\exists s \in w$: $s \geqq t'$, or $t'[X] \notin z[X]$. In each case, the corresponding relation in $POSS^*(p)$ must contain a tuple which subsumes t', however, $POSS^*(r) - POSS^*(q)$ contains a relation in which the corresponding relation does not. In the first case, the possibility of t' can be eliminated by the possibility of s in w that subsumes it, and in the second case, simply choose not to include t' in $POSS^*(r)$. Therefore, $POSS^*(p) \not\supseteq POSS^*(r) - POSS^*(q)$, which is a contradiction.

We conclude that null-extended difference is an adequate and restricted generalization of Δ-difference for *POSS** with respect to unnesting. □

6.2.3. Intersection, Cartesian Product, and Select. We will not formally define "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*, which uses our notion of equality of nulls and is otherwise identical to standard select.

6.2.4. Join. The problems involved in defining join operations for relations with nulls and for nested relations have been discussed before. Combining nulls with nested relations 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 27. 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 10. *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 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 adequacy and then restrictedness.

adequate: We show $POSS^*(r \rtimes^{e'} q) \supseteq POSS^*(r) \rtimes^{e} POSS^*(q)$. Let $p = r \rtimes^{e'} q$ and $\hat{p} \in POSS^*(r) \rtimes^{e} POSS^*(q)$. Also, let C be the common zero order attributes of r and q. Then, there must be $\hat{r} \in POSS^*(r)$ and $\hat{q} \in POSS^*(q)$ such that $\hat{p} = \hat{r} \rtimes^{e} \hat{q}$. Let t_p be a tuple in p. Then, there are tuples $t_r \in r$ and $t_q \in q$ such that $t_p[C] = t_r[C] = t_q[C]$. There are also tuples $t_{\hat{r}} \in \hat{r}$ and $t_{\hat{q}} \in \hat{q}$ that agree on C with t_p and will participate in the join giving $t_{\hat{p}}$. Now, the common higher order attributes X of $t_{\hat{r}}$ and $t_{\hat{p}}$ will participate in an extended intersection, the result of which will subsume the result of the null-extended intersection of $t_r[X]$ and $t_q[X]$. Therefore, $t_{\hat{p}} \geqq t_p$, $\hat{p} \geqq p$, and so $\hat{p} \in POSS^*(p)$.

restricted: We show that there does not exist p such that $POSS^*(r \rtimes^{e'} q) \supsetneq POSS^*(p) \supseteq POSS^*(r) \rtimes^{e} POSS^*(q)$. Suppose there is some p. If $POSS^*(r \rtimes^{e'} q) \supsetneq POSS^*(p)$, then there must be some tuple t in p that does not subsume any tuple in $r \rtimes^{e'} q$. Thus, t contains non-null values which must occur in any possibility of p, but not in all possibilities of $r \rtimes^{e'} q$. 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 $r \rtimes^{e'} q$, 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 values, yet there are possibilities of r and q which do not. Therefore, there is a possibility of r and q whose extended join is not a possibility of p. So, $POSS^*(p) \not\supseteq POSS^*(r) \rtimes^{e} POSS^*(q)$, 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 $POSS^*$. □

6.2.5. 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 28. The *null-extended projection* of relation r on attributes X is

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

Proposition 11. *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 8). □

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

Fig. 7. Relation satisfying $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow C$, but not $A \rightarrow\rightarrow C$ when **dne** nulls are not equated

7. Dependencies in a 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.

7.1. Non-existent Nulls

In [17], a sound and complete axiomatization for functional and multivalued dependencies is 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 (Fig. 7).

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* [5]. On examining the first two tuples in relation r, it must be true that there is an object involving attributes A, C, and D, and no subset of them. Otherwise, we would have to add two tuples matching the first two tuples in r but with the C and D values swapped. However, on examining the last four tuples, where **dne** nulls do not occur, there are independent AC and AD objects. If we accept this, then we must accept the fact that there are two different

semantics for tuples in r. If the value of B is **dne** then an ACD association must exist, and if the value is not **dne** then independent AC and AD associations must exist, in addition to associations involving B. We do not believe this is a plausible way to interpret a relation.

The solution is to equate **dne** nulls from the same domain, as we have done previously. 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 $\langle a_3, b, \textbf{dne}, d_3\rangle$ 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 $\pi_0(r)$ there do not exist two tuples t_1 and t_2 where $t_1[A] = \textbf{dne}$, $t_1[A] \neq t_2[A]$, and $t_1[O-A] = t_2[O-A]$, for any $A \in O$.

7.2. *Unknown Nulls*

The effect of **unk** nulls on functional dependencies has been adequately covered in [39]. 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 [16, 17] 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 Fig. 7, with **unk** nulls replacing the **dne** nulls. Since **unk** nulls are placeholders for actual facts about the world, the dependencies with which we have constrained the world are not altered by the presence of these nulls. Therefore, dependencies which follow from the given dependencies in a world without null values must still hold in a world with nulls. Thus, a relation such as r with **unk** nulls, must not be a complete or accurate representation of the world, since for any relation r, every relation in $POSS(r)$ must satisfy all FDs and MVDs which can be derived from the given dependencies.

7.3. *No-information Nulls*

The only published work dealing with dependencies and the no-information interpretation of nulls is an axiomatization of FDs by [2]. 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 [17], about the lack of transitivity in this model. Based on the lattice developed in Sect. 3, 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 and earlier statement, that the definitions of FD and MVD need not be changed as long as the convention that two values from the same extended domain are equal if they are the same value and neither one is **ni** or **unk**.

7.4. Join Dependency

At first glance, there does not seem to be any good way to define the join dependency on relations with nulls. Consider the tuple $\langle a,$ **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.

8. 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 nested database with a straightforward extension. We discovered that our extended operators for nested relations 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 nested 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 [30].

References

1. Abiteboul, S., Bidoit, N.: Non first normal form relations to represent hierarchically organized data. In: Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pp. 191–200. Waterloo, 1984
2. Atzeni, P., Morfuni, N.M.: Functional dependencies in relations with null values. Inf. Process. Lett. **18**, 233–238 (1984)
3. Biskup, J.: A formal approach to null values in database relations. In: Gallaire, H., Minker, J., Nicolas, J. (eds.) Advances in Database Theory, Volume 1, pp. 299–341. New York: Plenum Press 1981
4. Codd, E.F.: Extending the database relational model to capture more meaning. ACM Trans. Database Syst. **4**, 397–434 (1979)
5. Fagin, R., Mendelzon, A.O., , Ullman, J.D.: A simplified universal relation assumption and its properties. ACM Trans. Database Syst. **7**, 343–360 (1982)
6. Fischer, P.C., Van Gucht, D.: Determining when a structure is a nested relation. In: Proceedings of the Eleventh International Conference on Very Large Databases, pp. 171–180. Stockholm, 1985
7. Fischer, P.C., Van Gucht, D.: Weak multivalued dependencies. In: Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pp. 266–274. Waterloo, 1984
8. Güting, R.H., Zicari, R., Choy, D.M.: An Algebra for Structured Office Documents. Technical Report RJ 5559 (56648), IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, 1987
9. Grant, J.: Null values in a relational data base. Inf. Process. Lett. **6**, 156–157 (1977)
10. Jaeschke, G.: Nonrecursive Algebra for Relations with Relation Valued Attributes. Technical Report 84.12.001, Heidelberg Scientific Center. IBM Germany 1984
11. Jaeschke, G.: Recursive Algebra for Relations with Relation Valued Attributes. Technical Report 84.01.003, Heidelberg Scientific Center, IBM Germany 1984
12. Jaeschke, G., Schek, H.-J.: Remarks on the algebra of non first normal form relations. In: Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pp. 124–138. Los Angeles, 1982
13. Kambayashi, Y., Tanaka, K., Takeda, K.: Synthesis of unnormalized relations incorporating more meaning. Inf. Sci. **29**, 201–247 (1983)
14. Keller, A., Winslett Wilkins, M.: On the use of an extended relational model to handle changing incomplete information. IEEE Trans. Software Eng. **11**, 620–633 (1985)
15. Korth, H.F., Silberschatz, A.: Database System Concepts. New York: McGraw-Hill 1986
16. Lien, Y.E.: Multivalued dependencies with null values in relational data bases. In: Proceedings of the Fifth International Conference on Very Large Databases, pp. 61–66. Rio De Janeiro 1979
17. Lien, Y.E.: On the equivalence of database models. J. ACM **29**, pp. 333–362 (1982)

18. Lipski, W. Jr.: On databases with incomplete information. J. ACM **28**, 41–70 (1981)
19. Lipski, W. Jr.: On semantic issues connected with incomplete information databases. ACM Trans. Database Syst. **4**, 262–296 (1979)
20. Maier, D.: Discarding the universal relation assumption: Preliminary results. In: Proceedings of the XP1 Workshop on Relational Database Theory. New York 1980
21. Maier, D.: The Theory of Relational Databases. Rockville, MD: Computer Science Press 1983
22. Makinouchi, A.: A consideration on normal form of not-necessarily-normalized relation in the relational data model. In: Proceedings of the Third International Conference on Very Large Databases, pp. 447–453. Tokyo 1977
23. Imieliński, T., Lipski, W. Jr.: Incomplete information and dependencies in relational databases. In: Proceedings of ACM-SIGMOD 1983 International Conference on Management of Data, pp. 178–184. San Jose 1983
24. Imieliński, T., Lipski, W. Jr.: Incomplete information in relational databases. J. ACM **31**, 761–791 (1984)
25. Imieliński, T., Lipski, W., Jr.: On representing incomplete information in a relational database. In: Proceedings of the Seventh International Conference on Very Large Databases, pp. 388–397. Cannes 1981
26. Ozsoyoğlu, G., Ozsoyoğlu, Z.M.: An extension of relational algebra for summary tables. In: Proceedings of the 2nd International (LBL) Conference on Statistical Database Management, pp. 202–211. Los Angeles 1983
27. Ozsoyoğlu, Z.M., Yuan, L.-Y.: A new normal form for nested relations. ACM Trans. Database Syst. **12**, 111–136 (1987)
28. Pistor, P., Traunmüller, R.: A data base language for sets, lists, and tables. Inf. Syst. **11**, 323–336 (1986)
29. Roth, M.A.: Theory of Non-First Normal Form Relational Databases. PhD thesis, The University of Texas at Austin, Austin, Texas, May 1986
30. Roth, M.A., Kirkpatrick, J.E.: Algebras for nested relations. Data Eng. **11**, 39–47 (1988)
31. Roth, M.A., Korth, H.F., Batory, D.S.: SQL/NF: A query language for $\neg$1NF relational databases. Inf. Syst. **12**, 99–114 (1987)
32. Roth, M.A., Korth, H.F., Silberschatz, A.: Extended algebra and calculus for $\neg$1NF relational databases. ACM Trans. Database Syst. **13**, 389–417 (1988)
33. Schek, H.-J.: Towards a basic relational NF^2 algebra processor. In: Proceedings of the International Conference on Foundations of Data Organization, pp. 173–182. Kyoto 1985
34. Schek, H.-J., Scholl, M.H.: An Algebra for the Relational Model with Relation-Valued Attributes. Technical Report DVSI-1984-T1. Technical University of Darmstadt, Darmstadt, FRG 1984
35. Schek, H.-J., Scholl, M.H.: Die NF^2-relationenalgebra zur einheitlichen Manipulation externer, konzeptueller und interner Datenstrukturen. In: Schmidt, J. (ed.) Sprachen für Datenbanken. Informatik Fachberichte Nr. 72. Berlin Heidelberg New York Tokyo: Springer 1983
36. Scholl, M.H.: Theoretical foundation of algebraic optimization utilizing unnormalized relations. In: International Conference on Database Theory, Rome. Lect. Notes Comput. Sci. **243**, 380–396 (1986)
37. Sciore, E.: Null Values, Updates, and Normalization in Relational Databases. Technical Report, Department of Electrical Engineering and Computer Science, Princeton University 1979
38. Thomas, S.J., Fischer, P.C.: Nested relational structures. In: Kanellakis, P.C. (ed.) Advances in Computing Research, Vol. 3. The Theory of Databases, pp. 269–307. Greenwich, CT: JAI Press 1985
39. Vassiliou, Y.: Functional dependencies and incomplete information. In: Proceedings of the Sixth International Conference on Very Large Databases, pp. 260–269. Montreal 1980
40. Vassiliou, Y.: Null values in data base management: A denotational semantics approach. In: Proceedings of ACM-SIGMOD International Conference on Management of Data, pp. 162–169. Boston 1979
41. Wong, E.: A statistical approach to incomplete information in database systems. ACM Trans. Database Syst. **7**, 470–488 (1982)
42. Zaniolo, C.: Database relations with null values. J. Comput. Syst. Sci. **28**, 142–166 (1984)
43. Zaniolo, C.: A Formal Treatment of Nonexistent Values in Database Relations. Technical Report, Bell Laboratories, Holmdel, NJ, 1983

Received September 11, 1987 / April 12, 1989