Extended Algebra and Calculus for ¬1NF Relational Databases*

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

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

TR–84–36 December 1984
revised January 1985

* Former title: *Theory of Non-First-Normal-Form Relational Databases*

Extended Algebra and Calculus for ¬1NF Relational Databases

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

Department of Computer Science
University of Texas at Austin
Austin, TX 78712-1188

Abstract

Relaxing the assumption that relations are always in First-Normal-Form (1NF) necessitates a re-examination of the fundamentals of relational database theory. In this paper we take a first step towards unifying the various theories of ¬1NF databases. We start by determining an appropriate model to couch our formalisms in. We then define an extended relational calculus as the theoretical basis for our ¬1NF database query language. We define a minimum extended relational algebra and prove its equivalence to the ¬1NF relational calculus. We define a class of ¬1NF relations with certain "good" properties and extend our algebra operators to work within this domain. We prove certain desirable algebraic equivalences that hold only if we restrict our language to this domain.

1. Introduction

A fundamental property of existing relational database theory is that relations must be at least in first-normal-form (1NF), in which only atomic (i.e., non-decomposable) valued domains are allowed. Although there are many valid arguments in support of 1NF normalization it has been long recognized by both researchers and practitioners that there is a need for unnormalizing relations. Codd, himself, less than a year after the original paper [Cod1] was published, stated in [Cod2]:

> "For presentation purposes, it may be desirable to convert a normalized relation to unnormalized form. The operation of *factoring* accomplishes this."

This idea lay dormant until Makinouchi [Mak] recognized the need for utilizing relational databases for non-business data processing applications such as picture and map processing, and computer-aided design. He realized the restriction imposed by 1NF hinders such applications and proposed allowing set-valued domains and defined a new normal form which incorporated set-values in a 4NF design.

Exploratory work dealing with ¬1NF relations includes Furtado [Fur] and Furtado and Kerschberg [FK], who examined horizontal decomposition of relations and their properties, Orman [Orm], who looked at indexed (partitioned) relations as a conceptual model, and Kambayashi, et al. [KTT], who proposed using ¬1NF relations to handle redundancies caused by dependency constraints as opposed to using decomposition.

Jaeschke and Schek [JS] defined a class of ¬1NF relations limiting domains to be powersets of atomic domains. They extended the relational algebra by adding a nest operator to form sets of values and an unnest operator to take sets apart. Arisawa, et al. [AMM] examined ¬1NF relations in which all attributes of the corresponding 1NF relations are nested one level deep. Abiteboul and Bidoit [AB] describe the use of ¬1NF relations in the VERSO machine [Ban]. They allow multiple attributes to be nested and define recursive operators to work on their structures.

† Research partially supported by an IBM Faculty Development Award and NSF grant DCR-8507224

‡ Research partially supported by NSF grants DCR-8507224, MCS8104017, and MCS8122039

Özsoyoğlu and Özsoyoğlu [OO] extend the algebra of [JS] by adding aggregation-by-template, an operator to apply a function to a set of values. Aggregation was also studied by Klug [Klu] and Epstein [Eps]. Fischer and Thomas [FT] explore general ¬1NF relations as in [AB] but with the basic operators of [JS]. Finally, Jaeschke [Jae1, Jae2], and Schek and Scholl [Sch, ScS] define recursive relational algebras which are equivalent to the nonrecursive algebra of [FT].

It is now well recognized that non-traditional database applications (e.g., image and sound processing, text processing, and forms management) require relations to be in non-first-normal-form (¬1NF). Thus, in this paper we take a first step towards unifying the various theories of ¬1NF databases. We start by determining an appropriate model to couch our formalisms in. We then define a relational calculus as the theoretical basis for our ¬1NF database query language. We define a minimum relational algebra and prove its equivalence to the ¬1NF relational calculus. We define a class of ¬1NF relations with certain "good" properties and extend our algebra operators to work within this domain. We prove certain desirable algebraic equivalences that hold only if we restrict our language to this domain.

2. Model Definition

When set-valued domains are allowed, standard first-order predicate logic becomes cumbersome and somewhat inadequate to use. Several existing models have the extensions we desire, the database abstractions of Smith and Smith [SmS], the Format Model of Hull and Yap [HY], and the databases logics of Jacobs [Jac] and Kuper and Vardi [KV]. We follow the the lead of [FT] and adopt a formalism adapted from the database logic of Jacobs. The following condensation is basically that of [FT].

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}, $1 \leq i \leq n$, are names. R_j is a *higher order name* if it appears on the left hand side of some rule; otherwise it is *zero order*. Each zero order name has an associated *domain* from which the names values are drawn. 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, names on the right hand side of the same rule are unique, and to avoid ambiguity, no two rules can have the same name on the left hand side.

To illustrate this, consider an employee database containing, for each employee (EMP), his identification number (ID#), his name (E_NAME), information about his children (CH) and a salary history (S_H). A possible database scheme would consist of the rules:

$$\begin{aligned} \text{EMP} &= (\text{ID\#}, \text{E\_NAME}, \text{CH}, \text{S\_H}), \\ \text{CH} &= (\text{C\_NAME}, \text{ AGE}, \text{ SEX}), \\ \text{S\_H} &= (\text{YEAR}, \text{ SALARY}). \end{aligned}$$

In this example the higher order names are EMP, CH and S_H. All others are zero order names.

A name R_j is *external* if it appears *only* on the left hand side of some rule, otherwise it is *internal*. Thus in the above example, EMP is external while all other names are internal.

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

1. $R_j = (R_{j_1}, R_{j_2}, \ldots, R_{j_n})$ is in the subscheme, and

2. When a higher order name R_k is on the right hand side of some rule in the subscheme, the rule $R_k = (R_{k_1}, R_{k_2}, \ldots, R_{k_n})$ is also in the subscheme.

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

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

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

A 1NF database scheme is a collection of rules of the form $R_j = (R_{j_1}, R_{j_2}, \ldots, R_{j_n})$ where all the R_{j_i} are zero order. Our ¬1NF scheme may contain any combination of zero or higher order names on the right hand side of the rules as long as the scheme remains nonrecursive. Note that a nested relation is represented simply as a higher order name on the right hand side of a rule.

Let R be a name in a database scheme S. An *instance* of R, written r, is an ordered pair of the form $\langle R, V_R \rangle$ where V_R is a value for name R. When R is a zero order name, V_R is just any value from the domain of R. When R is a higher order name, V_R must be expanded in terms of the names on the right hand side of rule R. We write $t[A_1 A_2 \ldots A_n]$ to denote the $A_1, A_2, \ldots, A_n$ components of tuple t. A database structure $\mathcal{S} = \langle S, s \rangle$ refers to a database scheme S and some instance of that scheme s. A relation structure $\mathcal{R} = \langle R, r \rangle$ denotes a relation scheme R and instance r. In no case are null values or nested empty relations allowed in instances. For simplicity, we do not consider null values in this paper. [RKS] presents a thorough study of null values in ¬1NF relations.

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

In this paper, we use the terms attribute and name interchangeably, especially when it is not important to distinguish between zero and higher order names. If X and Y are sets of attributes then XY is the union of X and Y. We use A instead of $\{A\}$ when the meaning is clear from context.

3. Relational Calculus

3.1 Basic Definitions

Borrowing notation from [Ull], we define a tuple relational calculus (TRC) with expressions of the form $\{t \mid \psi(t)\}$, where t is a tuple variable of fixed length and ψ is a formula built from atoms and a collection of operators defined below. The arity (i) of a tuple variable t can be specified by $t^{(i)}$.

The atoms of formulas ψ are of four types.

1. $s \in r$, where s is a tuple variable, and r is a relation name. This specifies that s is a tuple in relation r, or s is an element of r. The arity of s is equal to the degree of r.
2. $s \in t[i]$ where t and s are tuple variables. This specifies that s is a tuple in the relation specified by the ith component of t, whose value must be a set-of-tuples. The arity of s is the arity of the tuples in the set.
3. $a\ \theta\ s[i]$, $s[i]\ \theta\ a$, $s[i]\ \theta\ t[j]$, where s and t are tuple variables, a is a constant, and θ is an arithmetic comparison operator $(=, >)$. Note that constants may be simple values or non-empty sets-of-values, however equality is the only operator which can compare non-simple values. We chose $=$ and $>$ since

r

A	B		E'
	C	D'	E
		D	
a_1	c_1	11	e_1
		2	e_2
	c_2	1	e_3
		3	
a_2	c_1	11	e_1
		14	e_3
		6	
	c_3	14	
		16	

A	B		E'
	C	D'	E
		D	
a_1	c_1	11	e_1
			e_2
			e_3
a_2	c_1	11	e_1
		14	e_3
	c_3	14	
		16	

Figure 3-1. Result of calculus query 4.

other comparison operators, such as $<$, $\supseteq$, $\subset$, etc., can be expressed with calculus expressions which do not use them.

4. $s[i] = \{u | \psi'(u, t_1, t_2, \ldots, t_k)\}$, where ψ' is a formula with free tuple variables $u, t_1, t_2, \ldots, t_k$; s is some t_j. This specifies that the ith attribute of s is the set of u tuples such that ψ' holds. Note, if no tuples u satisfy ψ' then this atom evaluates to false. This is to comply with our requirement that no null values appear in instances.

Formulas are defined with the usual operators ($\neg, \wedge, \vee, \forall, \exists$). See [Ull] for the formal definitions.

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

1. Given a 1NF relation r on scheme $R = (A, B)$, the TRC expression which nests r on the B attribute producing a relation with scheme $R' = (A, B'), B' = (B)$ is:

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

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

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

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

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

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

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

Figure 3-1 shows a sample relation r and the result of this query.

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

A *safe* TRC incorporating nesting is defined in the same manner as an ordinary safe TRC. The primary difference is that our attributes may be defined, not only over simple domains, but domains that may contain sets, sets-of-sets, and so on. Thus, we define $DOM(\psi)$ as the set of symbols that appear explicitly in expression ψ or are *recursively* components of some tuple in some relation mentioned in ψ. An expression ψ is safe if each component of any t that satisfies ψ must be a member of or, recursively, a relation on $DOM(\psi)$.

With this modification of $DOM(\psi)$, and the proviso that *each* calculus expression, nested or otherwise, must be safe, the rest of the formal definition of safety is the same as [Ull].

4. Relational Algebra

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

1. *Nest* takes a relation structure $\mathcal{R} = \langle R, r\rangle$ and aggregates over equal data values in some subset of the names in R. Formally, let R be a relation scheme, in database scheme S, which contains a rule $R = (A_1, A_2, \ldots, A_n)$ for external name R. Let $\{B_1, B_2, \ldots, B_m\} \subset E_R$ and $\{C_1, C_2, \ldots, C_k\} = E_R - \{B_1, B_2, \ldots, B_m\}$. 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 and $(B_1, B_2, \ldots, B_m)$ does not appear on the right hand side of any rule in S. Then $\nu_{B=(B_1,B_2,\ldots,B_m)}(\mathcal{R}) = \langle R', r'\rangle = \mathcal{R}'$ where:
   1. $R' = (C_1, C_2, \ldots, C_k, (B_1, B_2, \ldots, B_m)) = (C_1, C_2, \ldots, C_k, B)$ 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 $t[C_1C_2\cdots C_k] = u[C_1C_2\cdots C_k] \wedge$ $t[B] = \{v[B_1B_2\cdots B_m] \mid v \in r \wedge v[C_1C_2\cdots C_k] = t[C_1C_2\cdots C_k]\}\}$.
2. *Unnest* takes a relation structure nested on some set of attributes and disaggregates the structure to make it a "flatter" structure. Formally, let R be a relation scheme, in database scheme S, which contains a rule $R = (A_1, A_2, \ldots, A_n)$ for external name R. Assume B is some higher order name in E_R with an associated rule $B = (B_1, B_2, \ldots, B_m)$. Let $\{C_1, C_2, \ldots, C_k\} = E_R - B$. Then $\mu_{B=(B_1,B_2,\ldots,B_m)}(\mathcal{R}) = \langle R', r'\rangle = \mathcal{R}'$ where:
   1. $R' = (C_1, C_2, \ldots, C_k, B_1, B_2, \ldots, B_m)$ 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 $t[C_1C_2\cdots C_k] = u[C_1C_2\cdots C_k] \wedge t[B_1B_2\cdots B_m] \in u[B]\}$.

   Note that unnesting an empty set produces no tuples.

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

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

r

A	C	D	E
a_1	c_1	d_1	e_1
a_1	c_1	d_2	e_1
a_1	c_1	d_2	e_2
a_2	c_2	d_1	e_1
a_2	c_2	d_1	e_2

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

A	B		E'
	C	D	E
a_1	c_1	d_1	e_1
	c_1	d_2	
a_1	c_1	d_2	e_2
a_2	c_2	d_1	e_1
			e_2

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

A	B		E'
	C	D	E
a_1	c_1	d_1	e_1
a_1	c_1	d_2	e_1
			e_2
a_2	c_2	d_1	e_1
			e_2

Figure 4-1. Result of algebra query 2.

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

$$\nu_{B=(CD)}(r)$$

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

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

$$\nu_{B=(CD)}(\nu_{E'=(E)}(r)) \quad \text{or} \quad \nu_{E'=(E)}(\nu_{B=(CD)}(r))$$

Although both of these expressions produce the desired scheme, the relations may be radically different (see Figure 4-1).

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

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

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

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

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

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

$$\mu_{B}(s)$$

6. Given relation s on S as in 4, the relation with each of the D' sets within each B subrelation unnested, producing the relation with scheme $S' = (A, B, E'), B = (CD), E' = (E)$, is:

$$\nu_{B=(CD)}(\mu_{D'}(\mu_{B}(s)))$$

5. Equivalence of the Relational Calculus and the Relational Algebra

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

5.1 Reduction of Relational Algebra to Relational Calculus

Theorem 5-1. *If E is a relational algebra expression, then there is a safe expression in the relational calculus equivalent to E.*

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

Case 6: $E = \nu_{B=(A_1 A_2 \ldots A_k)}(E_1)$. Let E_1 be equivalent to safe expression $\{t^{(n)} \mid \psi_1(t)\}$ and let attribute A_i correspond to the j_i'th attribute, for $1 \leq i \leq k$, and let all attributes not among the A_i correspond to the j_ℓ'th attribute, for $k < \ell \leq n$. Then E is equivalent to

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

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

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

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

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

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

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

5.2 Reduction of Relational Calculus to Relational Algebra

Theorem 5-2. *If E is a safe expression in the relational calculus then there is a relational algebra expression equivalent to E.*

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

Lemma 5-1. *If ψ is any formula in tuple calculus then there is an equivalent formula ψ' of tuple calculus with no occurrences of $\wedge$ or $\forall$. If ψ is safe, so is ψ'.*

Proof: See [Ull], Lemma 5-2. □

Lemma 5-2. *If ψ is any formula in tuple calculus then there is an algebra expression for $DOM(\psi)$.*

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

Our proof of the theorem mirrors the proof in [Ull] of the equivalence of the (1NF) relational calculus and algebra. In [Ull], an algebra expression was created which produced a unary relation E of all values either mentioned explicitly as constants in the calculus expression or exists in any relation mentioned in the calculus expression. Each atom of the calculus expression is then translated as a function of $\times_{i=1}^{n} E$ where n is the number of attributes in all tuples variables being used in the subexpression where the atom occurs. The relation E is basically a *domain* of values from which the calculus expression must create the tuples in the result. However, when we move to ¬1NF relations, it is not possible to create a domain of values using this technique. Each tuple variable may range over values that are nested relations, and so to include all possible nested relations, we would have to have a technique for creating a powerset using the relational algebra. Since it is not possible to create a powerset using the algebra, we will use subsets of all possible tuples for each tuple variable and each component of a tuple variable that is defined as a nested relation. These *limited domains* contain all possible tuples from which the calculus expression will select tuples for the result.

Definition 5.1: A *limited domain* for a tuple variable t, denoted D_t, appearing in a safe calculus expression, $\{x \mid \psi(x)\}$, is an extended relational algebra expression which produces a ¬1NF relation r in which each nested relation in r, and r itself, contains all tuples, made up of values from $DOM(\psi)$, which need to be tested for inclusion or exclusion, by the atoms of the calculus expression referring to t.

If there is a subformula of the form $(\exists t)(p(t))$, then the limited domain for t contains tuples to be *included* in the result if they satisfy p. If there is a subformula of the form $\neg(\exists t)(p(t))$, or $(\forall t)(\neg p(t))$, then the limited domain for t contains tuples to be *excluded* from the result if they satisfy p. In the main body of the proof we present a way to construct an algebra expression which performs the proper inclusions and exclusions on the tuples in each limited domain.

Lemma 5-3. *Given a safe tuple calculus expression $\{t \mid \psi(t)\}$, there is an algebra expression D_{t_i} for the limited domain of each tuple variable t_i mentioned in ψ, or any nested expression of ψ.*

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

The following algorithm examines each atom in the expression and adds algebra expressions to each domain so that the possible values which that atom references will be included in the domain. The intuition behind this algorithm is as follows. When atoms refer to named relations and constants the reference is direct and known. However, when the atoms refer only to tuple variables, then the reference is indirect, and must be solved in terms of tuple variables which have direct and known references. In addition, there may be more than one atom which references a particular attribute of a tuple variable, and so we may get multiple expressions for each domain. Thus, as the algorithm creates the algebra expression for each domain, it also creates a graph which tells us how to solve the indirect references in our algebra expressions. Let D_t^i be the algebra expression for the limited domain of the ith attribute of tuple variable t. The graph will be constructed of nodes, directed edges, and directed and-edges. A directed and-edge is a single edge which goes

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

Algorithm 1

Create a graph with one node labeled RC, standing for named relations and constants;
For each tuple variable t
do
let k denote the arity of t;
create k nodes in the graph, labeled D_t^i, $1 \leq i \leq k$
end do
For each atom in the calculus expression
do
case atom
$t \in r$:
let k denote the arity of t;
add directed edges from D_t^i to RC, $1 \leq i \leq k$;
for $1 \leq i \leq k$, label the edge from D_t^i to RC with $\pi_i(r)$;
$t \in u[j]$:
let k denote the arity of t;
add directed edges from D_t^i to D_u^j, $1 \leq i \leq k$;
for $1 \leq i \leq k$, label the edge from D_t^i to D_u^j with $\pi_i(\mu_1(D_u^j))$;
$t[j]\,\theta\,a$ or $a\,\theta\,t[j]$, $\theta \in \{=, >\}$:
add a directed edge from D_t^j to RC;
label the edge C, where C is a new unary relation containing the single tuple $< a >$;
add a special label, Φ, to the edge if $\theta = >$;
$t[j]\,\theta\,u[\ell]$, $\theta \in \{=, >\}$:
add directed edges from D_t^j to D_u^ℓ and from D_u^ℓ to D_t^j;
label the edge from D_t^j to D_u^ℓ with D_u^ℓ;
label the edge from D_u^ℓ to D_t^j with D_t^j;
add a special label, Φ, to each edge if $\theta = >$;
$t[j] = \{u^{(\ell)} \mid \psi'(u)\}$:
add a directed and-edge from D_t^j to the set of nodes D_u^i, $1 \leq i \leq \ell$;
label the and-edge $\nu_{1=(1,2,\ldots,\ell)}(D_u^1 \times D_u^2 \times \cdots \times D_u^\ell)$;
end case
end do
Mark node RC;

Let $\mathcal{D}$ be the algebra expression for $DOM(\psi)$;
While some node in the graph is not marked
do
Choose an unmarked node N with at least one edge, without a special label Φ,
directed towards a marked node, or
at least one and-edge directed towards a set of nodes, all of which are marked, or
if neither of the above cases applies, at least one edge, with a special label Φ,
directed towards a marked node;
If the special case using label Φ was invoked then let $C = \mathcal{D}$ else let $C = \emptyset$;
Set the algebra expression for the domain labeled N to $L_1 \cup L_2 \cup \cdots \cup L_p \cup C$,
where p is the number of edges and and-edges directed from N to marked nodes,
and L_i is the label of the ith such edge;
Mark node N
end do
For each tuple variable t with arity k, set D_t to $D_t^1 \times D_t^2 \times \cdots \times D_t^k$.

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

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

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

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

Thus, either the expression is unsafe, or we have included all the necessary tuples in our limited domains,

and so the algorithm is correct. □

Proof of Theorem 5-2: Let $\{t \mid \psi(t)\}$ be a safe tuple calculus expression. We construct an equivalent algebra expression. By Lemma 5-3 we have an algebra expression D_x for each tuple variable x mentioned in ψ. By Lemma 5-1 we may assume that ψ has only the operators $\vee$, $\neg$, and $\exists$. We prove by induction on the number of operators in a subformula ω of ψ that if ω has free variable s, then

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

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

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

Since ψ is safe, intersection with D_t does not change the relation denoted, so we shall have proved the theorem.

In order to avoid problems where $\nu_A(\mu_A(r)) \neq r$ we assume each database relation $(r, q, \ldots)$, and their nested relations have an implicit keying attribute whose value uniquely determines the values of all other attributes. We consider this attribute to be added to each relation before it is used and removed when the relation is projected or presented as the final result, using appropriate algebra operations.

We now proceed with the inductive proof.

Basis: Zero operators in ω. Then ω is an atom, which we may take to be in one of the forms described in section 3.1. In order to specify an algebra expression for these atoms, which may, as themselves, specify infinite relations, we need to operate on an expression $D = D_{s_1} \times D_{s_2} \times \cdots \times D_{s_n}$, where the s_i are *all* free tuple variables of the formula ω of which this atom is currently a part.
The atoms are thus translated:

1. $s \in r$: Replace D_s in D by r.

2. $s \in t[i]$: Let $p_1, p_2, \ldots, p_k$ be the attributes of D_s in D, let $q*$ be the ith attribute of D_t in D, and let $q_1, q_2, \ldots, q_k$ be the attributes of $q*$.
   Then the desired expression is

   $$\nu_{q*=(q_1,q_2,\ldots,q_k)}(\sigma_{p_1=q_1 \wedge \cdots \wedge p_k=q_k}(\mu_{q*}(D)))$$

   By unnesting we can access the elements of $t[i]$ using standard relational algebra operators. The selection picks out those values corresponding to tuple variable s's domain D_s. The final nest returns the structure to its intended form.

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

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

4. $s[i] = \{u^{(j)} \mid \psi'(u, t_1, t_2, \ldots, t_n)\}$: We have s as one of $t_1, t_2, \ldots, t_n$, and j as the arity of a new tuple variable u. Let E' be an algebra expression for ψ', p be the ith attribute of D_s, and k be the arity of D, then the desired algebra expression is

$$\pi_{1,2,\ldots,k}(\sigma_{p=(k+1)}(\nu_{k+1=(k+1,k+2,\ldots,k+j)}(E')))$$

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

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

$$D \cap \{ t_1, t_2, \ldots, t_n \mid \omega_1(t_1, t_2, \ldots, t_n) \}$$

and E_2 an algebraic expression for

$$D \cap \{ t_1, t_2, \ldots, t_n \mid \omega_2(t_1, t_2, \ldots, t_n) \}$$

Then the desired expression is

$$E_1' \cup E_2'$$

.

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

$$D \cap \{ t_1, t_2, \ldots, t_n \mid \omega_1(t_1, t_2, \ldots, t_n) \}$$

then

$$D - E_1$$

is an expression for

$$D - \{ t_1, t_2, \ldots, t_n \mid \omega_1(t_1, t_2, \ldots, t_n) \}$$

which is equivalent to

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

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

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

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

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

which completes the induction, and proves the theorem. □

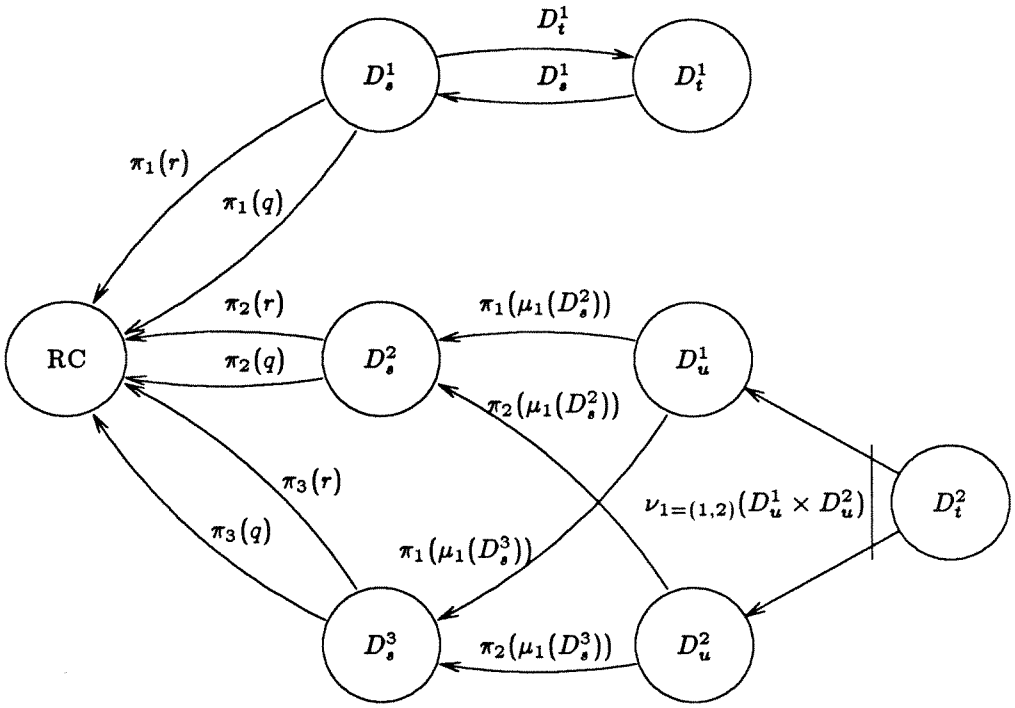

Figure 5-1. Graph produced by Algorithm 1.

5.3 Examples

To illustrate Lemma 5-2, consider the following calculus expression:

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

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

Before the marking phase of the algorithm the graph is as shown in Figure 5-1. During the marking phase, RC is marked. Then D_s^1, D_s^2, and D_s^3 are marked and the term D_t^1 is not included in the expression for D_s^1, since D_t^1 is not yet marked. Then, D_t^1, D_u^1, and D_u^2 can be marked, and, finally, D_t^2 is marked since all nodes at the end of the and-edge are marked. The algebra expressions at the end of the marking phase are:

$$D_s^1 = \pi_1(r) \cup \pi_1(q)$$
$$D_s^2 = \pi_2(r) \cup \pi_2(q)$$
$$D_s^3 = \pi_3(r) \cup \pi_3(q)$$
$$D_t^1 = D_s^1$$
$$D_t^2 = \nu_{1=(1,2)}(D_u^1 \times D_u^2)$$
$$D_u^1 = \pi_1(\mu_1(D_s^2)) \cup \pi_1(\mu_1(D_s^3))$$
$$D_u^2 = \pi_2(\mu_1(D_s^2)) \cup \pi_2(\mu_1(D_s^3))$$

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

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

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

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

For a complete example of the transformation process of Theorem 5-2, consider the following calculus expression:

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

where r is a relation on $R = (COURSE, DATE)$, $DATE = (MONTH, YEAR)$. This query is asking for all courses and the set of dates for the course with a year at most 1970.

Using the methodology of section 5.2 we translate this TRC expression into an equivalent relational algebra expression. We start by transforming the expression so that $\neg$, $\vee$, and $\exists$ are the only operators present.

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

The domains corresponding to each tuple variable are

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

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

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

We now proceed with the translation.

Translate each atom:

$$\begin{array}{lll} s \in r & \rightarrow & E_1 = D_t \times r \\ t[1]{=}s[1] & \rightarrow & E_2 = \sigma_{1=3}(D_t \times D_s) \\ t[2]{=}\{\ldots\} & \rightarrow & E_3 = \pi_{1,2,3,4}(\sigma_{2=5}(\nu_{5=(5,6)}(E'))) \end{array}$$

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

Translate negation and disjunction:

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

Translate existential quantifier and the final expression is:

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

E' is determined similarly.

Translate the atoms:

$$\begin{array}{lll} u \in s[2] & \rightarrow & E'_1{=}\nu_{4=(4,5)}(\sigma_{4=6 \wedge 5=7}(\mu_4(D_t \times D_s \times D_u))) \\ u[2] > \text{'1970'} & \rightarrow & E'_2{=}\sigma_{6>\text{'1970'}}(D_t \times D_s \times D_u) \end{array}$$

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

S_1

S_NAME	COURSE	
	C_NAME	GRADE
Jones	Math	A
	Science	B
Smith	Math	A
	Physics	C
	Science	A

S_2

S_NAME	COURSE	
	C_NAME	GRADE
Jones	Physics	B
Smith	Chemistry	A
	English	B

Figure 6-1. Two STUDENT instances.

$$E' = (D_t \times D_s \times D_u) - (((D_t \times D_s \times D_u) - E'_1) \cup E'_2)$$

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

$$\nu_{DATE}(\sigma_{YEAR \leq '1970'}(\mu_{DATE}(r)))$$

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

6. Restricting the class of ¬1NF relations

Consider the relation scheme

$$\text{STUDENT} = (\text{S\_NAME}, \text{COURSE})$$
$$\text{COURSE} = (\text{C\_NAME}, \text{GRADE})$$

In Figure 6-1 we have two instances of STUDENT, S_1 and S_2, where S_1 contains previous work of two students and S_2 contains some new data on these students.

A natural step would be to add the new information in S_2 to that in S_1. If we apply the union operator then we get the relation in Figure 6-2.

Although all of the information is certainly represented in this relation it lacks the intuitive appeal of the relation in Figure 6-3 in which the COURSE sets are combined for each unique value of STUDENT. One alternative is to use an unnest operation followed by the corresponding nest operation after taking the union. So the query would be

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

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

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

Definition 6.1: 6-1: Let $X \subseteq E_R$, $Y \subseteq E_R$ for some relation structure $\mathcal{R} = \langle R, r \rangle$. The *functional dependency* (FD), $X \rightarrow Y$, holds in r iff for all tuples t_1, t_2 in r if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. If X or Y is a higher order name then we mean set equality.

$S_1 \cup S_2$

S_NAME	COURSE	
	C_NAME	GRADE
Jones	Math	A
	Science	B
Jones	Physics	B
Smith	Math	A
	Physics	C
	Science	A
Smith	Chemistry	A
	English	B

Figure 6-2. Union of instances in Figure 6-1.

S_NAME	COURSE	
	C_NAME	GRADE
Jones	Math	A
	Science	B
	Physics	B
Smith	Math	A
	Physics	C
	Science	A
	Chemistry	A
	English	B

Figure 6-3. Better representation of Figure 6-2.

Definition 6.2: 6-2: Let $\mathcal{R} = \langle R, r \rangle$ be a relation structure with attribute set E_R containing zero order names $A_1, A_2, \ldots, A_k$ and higher order names $X_1, X_2, \ldots, X_\ell$. $\mathcal{R}$ is in *partitioned normal form* (PNF) iff

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

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

PNF is not normalization in the usual sense. Although PNF relations have less redundancy than their non-PNF counterparts, some nesting schemes will be less redundant than others, even if all relations are in PNF. PNF is a desirable goal for the representation of relationships in $\neg$1NF relations. This stems from our belief that a particular nesting scheme should not be used unless the FDs which enforce PNF hold in the relation. For a traditional normalization technique for $\neg$1NF relations, the reader is refered to [OY].

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

Theorem 6-1. *The class of PNF relations is closed under unnesting.*

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

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

The attributes of $\mathcal{R}'$ are, by definition of unnest, the attributes $(E_R - B) \cup (B_1 B_2 \cdots B_n)$. These attributes can be partitioned into four sets, the zero order names of E_R $(A_1 A_2 \cdots A_n)$, the higher order names in $E_R - B$ $(X_1 X_2 \cdots X_m)$, the zero order names of E_B $(B_1 B_2 \cdots B_\ell)$, and the higher order names of E_B $(Y_1 Y_2 \cdots Y_p)$. Our task then is to show that for any tuples t_1 and t_2, if $t_1[A_1 A_2 \cdots A_n B_1 B_2 \cdots B_\ell] = t_2[A_1 A_2 \cdots A_n B_1 B_2 \cdots B_\ell]$ then $t_1[X_1 X_2 \cdots X_m Y_1 Y_2 \cdots Y_p] = t_2[X_1 X_2 \cdots X_m Y_1 Y_2 \cdots Y_p]$.

Since $A_1 A_2 \cdots A_n \rightarrow X_1 X_2 \cdots X_m$ in $\mathcal{R}$, and unnesting only duplicates these values, we have that $t_1[X_1 X_2 \cdots X_m] = t_2[X_1 X_2 \cdots X_m]$. Since t_1 and t_2 agree on $A_1 A_2 \cdots A_n$, they came from the same tuple of r, and in this tuple $B_1 B_2 \cdots B_\ell \rightarrow Y_1 Y_2 \cdots Y_p$. So in the set of tuples obtained after unnesting the same FD applies and since t_1 agrees with t_2 on $B_1 B_2 \cdots B_\ell$, $t_1[Y_1 Y_2 \cdots Y_p] = t_2[Y_1 Y_2 \cdots Y_p]$. □

Theorem 6-2. *The nesting of a PNF relation is in PNF iff in the PNF relation* $\mathcal{R} = \langle R, r \rangle$, $A_1 A_2 \cdots A_k \rightarrow X_1 X_2 \cdots X_\ell$, *where* $A_1, A_2, \ldots, A_k$ *are the zero order names in* E_R *not being nested and* $X_1, X_2, \ldots, X_\ell$ *are*

the higher order names in E_R not being nested.

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

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

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

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

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

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

Case 5: $m > 0$, $n > 0$, $k < n$, $\ell < m$. Then we are nesting some zero order and some higher order names. So $A_1A_2\cdots A_k \rightarrow X_1X_2\cdots X_\ell$. Then during nesting a partition on $A_1A_2\cdots A_kX_1X_2\cdots X_\ell$ will be created and by definition each set X_0 of $A_{k+1}A_{k+2}\cdots A_nX_{\ell+1}X_{\ell+2}\cdots X_m$ values will be uniquely determined by $A_1A_2\cdots A_kX_1X_2\cdots X_\ell$. Thus, $A_1A_2\cdots A_k \rightarrow X_0X_1X_2\cdots X_\ell$. In each new subrelation the $A_{k+1}A_{k+2}\cdots A_n$ values are unique since $A_1A_2\cdots A_k$ was the same for each of these tuples and $A_1A_2\cdots A_n$ values were unique as $\mathcal{R}$ is in PNF. Thus $A_{k+1}A_{k+2}\cdots A_n \rightarrow X_{\ell+1}X_{\ell+2}\cdots X_m$ in each subrelation. Thus the relation is in PNF.

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

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

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

7. Extending the basic relational algebra operators

As the example in section 6 showed we need to extend our basic algebra operators to work within the class of PNF relations. We first extend the traditional set operators—union, intersection, difference, and cartesian product, and then extend natural join and projection. Some of these operators are similar to the extended operators of [AB]. However, our definitions arose out of the PNF requirement and since our model does not include null values or empty sets, the operations are well defined. In [AB], empty sets are allowed but null values are not, so there are problems when tuples with empty sets are unnested. Unlike [AB], we do not extend selection in this paper. Finally, we show that the class of PNF relations is closed under the

r_1

A	X		
	B	Y	
		C	D
a_1	b_1	c_1	d_1
		c_1	d_2
	b_2	c_1	d_1
		c_2	d_2
a_2	b_1	c_1	d_1
		c_2	d_3
		c_3	d_1
	b_3	c_2	d_2
a_3	b_4	c_1	d_2

r_2

A	X		
	B	Y	
		C	D
a_1	b_1	c_1	d_1
		c_3	d_2
	b_3	c_4	d_4
a_2	b_3	c_2	d_2
a_4	b_2	c_1	d_2
		c_1	d_3

$r_1 \cup^e r_2$

A	X		
	B	Y	
		C	D
a_1	b_1	c_1	d_1
		c_1	d_2
		c_3	d_2
	b_2	c_1	d_1
		c_2	d_2
	b_3	c_4	d_4
a_2	b_1	c_1	d_1
		c_2	d_3
		c_3	d_1
	b_3	c_2	d_2
a_3	b_4	c_1	d_2
a_4	b_2	c_1	d_2
		c_1	d_3

$r_1 \cap^e r_2$

A	X		
	B	Y	
		C	D
a_1	b_1	c_1	d_1
a_2	b_3	c_2	d_2

$r_1 -^e r_2$

A	X		
	B	Y	
		C	D
a_1	b_1	c_1	d_2
	b_2	c_1	d_1
		c_2	d_2
a_2	b_1	c_1	d_1
		c_2	d_3
		c_3	d_1
a_3	b_4	c_1	d_2

Figure 7-1. Examples of $\cup^e$, $\cap^e$, $-^e$.

extended operators. Note, the extended operators can be applied to to non-PNF relations in a well defined way, however, the result is not necessarily a PNF relation.

7.1 Traditional set operators

We present below our definitions of the extended traditional set operators and provide illustrative examples in Figure 7-1.

•**Extended Union**— In order to take the union of two relation structures $\mathcal{R}_1$ and $\mathcal{R}_2$ we require that they have equal relation schemes. The scheme of the resultant structure is also equal to schemes R_1 and R_2. We define union at the instance level as follows. Let X signify a zero order name in E_{R_1} and Y signify a higher order name in E_{R_1}. Then,

$$r_1 \cup^e r_2 = \{t \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 : (\forall X, Y \in E_{R_1} : t[X] = t_1[X] = t_2[X] \wedge t[Y] = (t_1[Y] \cup^e t_2[Y])))$$
$$\vee\ (t \in r_1 \wedge (\forall t' \in r_2 : (\forall X \in E_{R_1} : t[X] \neq t'[X])))$$
$$\vee\ (t \in r_2 \wedge (\forall t' \in r_1 : (\forall X \in E_{R_1} : t[X] \neq t'[X])))\}$$

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

•**Extended Intersection** has the same requirements and note about recursion as union. Two tuples intersect if they agree on their zero order names and they have non-empty extended intersections of their higher order names. Let X signify a zero order name in E_{R_1} and Y signify a higher order name in E_{R_1}. Then,

$$r_1 \cap^e r_2 = \{t \mid (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 :$$
$$(\forall X, Y \in E_{R_1} : t[X] = t_1[X] = t_2[X] \wedge t[Y] = (t_1[Y] \cap^e t_2[Y]) \wedge t[Y] \neq \emptyset))\}$$

•**Extended Difference** also has the same requirements and note about recursion as union. In $r_1 - r_2$ a tuple is retained from r_1 if it does not agree with any tuple in r_2 on the zero order names or if it does then it has non-empty extended differences between the higher order names. Let X signify a zero order name in E_{R_1} and Y signify a higher order name in E_{R_1}. Then,

$$r_1 -^e r_2 = \{t \mid (\exists t_1 \in r_1 \wedge \exists t_2 \in r_2 \wedge \exists Y \in E_R : \\ (\forall X, Y \in E_{R_1} : t[X] = t_1[X] = t_2[X] \wedge t[Y] = (t_1[Y] -^e t_2[Y]) \wedge t[Y] \neq \emptyset)) \\ \vee (t \in r_1 \wedge (\forall t' \in r_2 : (\forall X \in E_{R_1} : t[X] \neq t'[X])))\}$$

•**Cartesian Product** requires no changes to the basic definition. We just need to ensure that common names are changed to maintain uniqueness in the new structure. See [FT] for a formal definition.

7.2 Natural Join and Projection

We present below our definitions of the extended natural join and extended projection operators and provide illustrative examples in Figure 7-2.

•**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 could 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.

Let $\mathcal{R}_1 = \langle R_1, r_1 \rangle$ and $\mathcal{R}_2 = \langle R_2, r_2 \rangle$, be two relation structures. We define the extended natural join $\mathcal{R}_1 \bowtie^e \mathcal{R}_2$ as a recursive application of a rule similar to the definition of natural join used for standard flat 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 names is not empty.

Let X be the higher order names in $E_{R_1} \cap E_{R_2}$, $A = E_{R_1} - X$, and $B = E_{R_2} - X$. Then the join is $\mathcal{R}_1 \bowtie^e \mathcal{R}_2 = \langle R, r \rangle$ 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, the intersection specified in the instance definition is the extended intersection as defined above.

•**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 names left in the projected relation. Formally, we define *extended projection* as

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

This operation can also be logically accomplished by unnesting and nesting one higher order name in the projected relation. Note, that projection still removes duplicate tuples, that is those which agree on all attributes, with set equality holding on higher order names.

s_1

A	B	X	
		C	D
a_1	b_1	c_1	d_1
		c_2	d_2
		c_1	d_3
a_2	b_1	c_3	d_1
		c_2	d_2
		c_1	d_1
a_2	b_2	c_1	d_2
		c_3	d_2

s_2

E	B	X	
		C	D
e_1	b_1	c_1	d_1
		c_1	d_3
		c_3	d_4
e_3	b_2	c_3	d_2
e_4	b_1	c_3	d_1
		c_4	d_2

$s_1 \bowtie^e s_2$

A	E	B	X	
			C	D
a_1	e_1	b_1	c_1	d_1
			c_1	d_3
a_2	e_1	b_1	c_1	d_1
a_2	e_4	b_1	c_3	d_1
a_2	e_3	b_2	c_3	d_2

$\pi^e_{A,X}\ (s_1 \bowtie^e s_2)$

A	X	
	C	D
a_1	c_1	d_1
	c_1	d_3
a_2	c_1	d_1
	c_3	d_1
	c_3	d_2

$\pi^e_{E,B,X}\ (s_1 \bowtie^e s_2)$

E	B	X	
		C	D
e_1	b_1	c_1	d_1
		c_1	d_3
e_4	b_1	c_3	d_1
e_3	b_2	c_3	d_2

Figure 7-2. Examples of $\bowtie^e$, π^e.

7.3 Closure of PNF relations under the extended operators

Theorem 7-1. *The class of PNF relations is closed under extended union, extended intersection, extended difference, cartesian product, extended natural join, extended projection, and selection.*

Proof: The proofs for each operator are presented below.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8. Properties of the Extended Algebra

In addition to properties which hold between relational algebra expressions when using the usual realtional algebra operators there are some important equivalences which hold when using the extended operators defined in section 7.

Below we state some equivalences already proven on the extended algebra without the extended operators.

Theorem 8-1 ([FT]). *Given relation structures $\mathcal{R}$ and S the following properties hold*

a. $\mu_A(\nu_A(\mathcal{R})) = \mathcal{R}$

b. $\mu_A(\mu_B(\mathcal{R})) = \mu_B(\mu_A(\mathcal{R}))$

c. $\mu_A(\mathcal{R} \times S) = \mu_{A'}(\mathcal{R}) \times S$, if A is A' in $\mathcal{R}$
$\mu_A(\mathcal{R} \times S) = \mathcal{R} \times \mu_{A''}(S)$, if A is A'' in S

d. $\mu_A(\mathcal{R} \cup S) = \mu_A(\mathcal{R}) \cup \mu_A(S)$

e. $\nu_{B=(Y)}(\sigma_F(\mathcal{R})) = \sigma_F(\nu_{B=(Y)}(\mathcal{R}))$
$\mu_{B=(Y)}(\sigma_F(\mathcal{R})) = \sigma_F(\mu_{B=(Y)}(\mathcal{R}))$
where predicate F does not involve names B or Y.

f. $\mu_{B=(Y)}(\pi_{XB}(\mathcal{R})) = \pi_{XY}(\mu_{B=(Y)}(\mathcal{R}))$

Proof: See [FT]. □

The following theorem shows equivalences that hold for the extended operators.

Theorem 8-2. *Given relation structures $\mathcal{R}$ and S the following properties hold*

a. $\mu_A(\mathcal{R} \cap^e S) = \mu_A(\mathcal{R}) \cap^e \mu_A(S)$

b. $\mu_A(\mathcal{R} \bowtie^e S) = \mu_A(\mathcal{R}) \bowtie^e \mu_A(S)$

Proof (a): $\mu_A(\mathcal{R} \cap^e S) = \mu_A(\mathcal{R}) \cap^e \mu_A(S)$

Let $\mathcal{R} = \langle R, r\rangle$, $S = \langle R, s\rangle$, $A \in E_R$ and $A = (A_1, A_2, \ldots, A_m)$. The relation schemes for each side of the equivalence are equal since extended intersection preserves the scheme of the operands and so the unnest replaces A by $A_1A_2\cdots A_m$ on each side. We now show inclusion both ways to prove equivalence at the instance level.

$\subseteq$ We partition $\mu_A(\mathcal{R} \cap^e S)$ on $E_R - A_1A_2\cdots A_m$ and then show that all tuples $t_1, t_2, \ldots, t_n$ in any partition are in $\mu_A(\mathcal{R}) \cap^e \mu_A(S)$. The tuples $t_1, t_2, \ldots, t_n$ must have been unnested from a set of tuples $u_1, u_2, \ldots, u_k$ which form a partition on $E_R - A$ in $r \cap^e s$, where for all i, $1 \le i \le n$, there exists j, $1 \le j \le k$ such that $t_i[A_1A_2\cdots A_m] \in u_j[A]$. We then have $\cup_{j=1}^k u_j[A] = \{t_i[A_1A_2\cdots A_m] \mid 1 \le i \le n\}$. Each u_j was created by an extended intersection of two tuples, u_j^1 in r, and u_j^2 in s. So we have $\cup_{j=1}^k u_j[A] \subseteq \cup_{j=1}^k u_j^1[A]$ and $\cup_{j=1}^k u_j[A] \subseteq \cup_{j=1}^k u_j^2[A]$. When we unnest r on A the tuples u_j^1 unnest into tuples w_ℓ^1, $1 \le \ell \le p$. Similarily, when we unnest s on A the tuples u_j^2 unnest into tuples w_ℓ^2, $1 \le \ell \le q$, where $\cup_{j=1}^k u_j^1[A] = \{w_\ell^1[A_1A_2\cdots A_m] \mid 1 \le \ell \le p\}$ and $\cup_{j=1}^k u_j^2[A] = \{w_\ell^2[A_1A_2\cdots A_m] \mid 1 \le \ell \le q\}$. Following the chain of equalities we have $\{t_i[A_1A_2\cdots A_m] \mid 1 \le i \le n\} \subseteq \{w_\ell^1[A_1A_2\cdots A_m] \mid 1 \le \ell \le p\}$ and $\{t_i[A_1A_2\cdots A_m] \mid 1 \le i \le n\} \subseteq \{w_\ell^2[A_1A_2\cdots A_m] \mid 1 \le \ell \le q\}$. Therfore, in $\mu_A(r)$ and $\mu_A(s)$ we have agreement on at least all values of $A_1, A_2, \ldots, A_m$ in $\{t_i[A_1A_2\cdots A_m] \mid 1 \le i \le n\}$ and so $\mu_A(r) \cap^e \mu_A(s)$ includes all tuples $t_1, t_2, \ldots, t_n$.

$\supseteq$ We show that if $t \in (\mu_A(r) \cap^e \mu_A(s))$ then $t \in \mu_A(r \cap^e s)$. So t must be the extended intersection of some t_1 in $\mu_A(r)$ and some t_2 in $\mu_A(s)$. Also t_1 was unnested from some u_1 in r and t_2 was unnested from some u_2 in s. We have that $t_1[A_1A_2\cdots A_m] \in u_1[A]$ and $t_2[A_1A_2\cdots A_m] \in u_2[A]$. Therfore in the extended intersection of r and s, u_1 and u_2 will combine to produce w where, in particular, $w[A] = u_1[A] \cap^e u_2[A]$. w will then unnest to include t and we have that $t \in \mu_A(r \cap^e s)$. □

Proof (b): $\mu_A(\mathcal{R} \bowtie^e S) = \mu_A(\mathcal{R}) \bowtie^e \mu_A(S)$

This proof is similar to that of part a. □

In the next theorem we prove that nest is an inverse for unnest when operating on PNF relations.

Theorem 8-3. *Given relation structures $\mathcal{R}$, S in PNF the following properties hold*

a. $\nu_A(\mu_A(\mathcal{R})) = \mathcal{R}$

b. $\nu_{A_1}(\cdots(\nu_{A_n}(\mu_{A_n}(\cdots(\mu_{A_1}(\mathcal{R}))\cdots))\cdots) = \mathcal{R}$

Proof (a): $\nu_A(\mu_A(\mathcal{R})) = \mathcal{R}$

Let $\mathcal{R} = \langle R, r\rangle$, $A \in E_R$ and $A = (A_1 A_2 \cdots A_m)$. The relation schemes will be equal as the modification made by the unnest will be exactly reversed by nest. We now show inclusion both ways to prove equivalence at the instance level.

$\subseteq$ We show that if $t \in \nu_A(\mu_A(r))$ then $t \in r$. So there must be tuples $t_1, t_2, \ldots, t_n$ in $\mu_A(r)$ that form some partition of $\mu_A(r)$ on $E_R - A_1 A_2 \cdots A_m$, where $t[A] = \{t_i[A_1 A_2 \cdots A_m] \mid 1 \leq i \leq n\}$. We claim that in r, the tuple that unnested to $t_1, t_2, \ldots, t_n$, must be t. If it was not then there is some t' that also unnested, and agrees with $t_1, t_2, \ldots, t_n$ on $E_R - A_1 A_2 \cdots A_m$, since, by Theorem 6-1, $\mu_A(r)$ is in PNF. But t' was not included in the nest so must disagree on $E_R - A_1 A_2 \cdots A_m$ with $t_1, t_2, \ldots, t_n$. This is a contradiction, so $t \in r$.

$\supseteq$ We show that if $t \in r$ then $t \in \nu_A(\mu_A(r))$. Since r is in PNF $X \rightarrow A$. $t \in r$ implies $t_1, t_2, \ldots, t_n \in \mu_A(r)$ and these tuples are the only tuples which have equal X values due to $X \rightarrow A$. Therefore, in $\nu_A(\mu_A(r))$, $t_1, t_2, \ldots, t_n$ will reform to the tuple t again. □

Proof (b):

$\nu_{A_1}(\cdots(\nu_{A_n}(\mu_{A_n}(\cdots(\mu_{A_1}(\mathcal{R}))\cdots))\cdots) = \mathcal{R}$

This follows immediately from Theorem 6-1 and Theorem 8-3a. □

9. Conclusion

Some straightforward extensions to fundamental calculus and algebra languages allow us to have set-valued domains in the relational model. We lose none of the power of the original model and gain the power to manipulate sets of values. We have found that ¬1NF relations in PNF have some good properties, and by extending our basic relational algebra operators we can formulate queries that have more meaning to us. By limiting the database to PNF relations, we will always have that some sequence of nest operations will be an inverse for a sequence of unnest operations.

We plan to further investigate query languages for ¬1NF databases, especially towards more user friendly languages. We will examine the optimization and performance issues associated with set-valued domains. We will also look at dependency theory, as have others [FvG1, FvG2, KTT, TFS], to see what effect our extensions will have.

Some interesting problems occur in our formulations if we allow the empty set and null values, not the least of which is determining an appropriate semantics for the empty set, and the effect of unnesting on it. This will be the topic of a forthcoming report.

We also plan to implement our ideas within the ROSI project [KS] at the University of Texas at Austin.

10. Bibliography

[AB] Abiteboul, S. and N. Bidoit, "Non First Normal Form Relations to Represent Hierarchically Organized Data," *Proceedings of the ACM Symposium on Principles of Database Systems*, Waterloo (April 1984), 191–200.

[AMM] Arisawa, H., K. Moriya and T. Miura, "Operations and the Properties on Non-First-Normal-Form Relational Databases," *Proceedings of the Ninth International Conference on Very Large Databases*, Florence (October 1983), 197-204.

[Ban] Bancilhon, F., et al., "Verso : A Relational Back End Data Base Machine," *Proceedings of the International Workshop on Database Machines*, San Diego (1982).

[Cod1] Codd, E., "A Relational Model for Large Shared Data Banks," *Communications of the ACM 13*, 6 (June 1970), 377-387.

[Cod2] Codd, E., "Relational Completeness of Data Base Sublanguages." In *Courant Computer Science Symposium 6 on Data Base Systems*, ed. R. Rustin, New York (1971), 65–98.

[Eps] Epstein, R., "Techniques for processing of aggregates in relational database systems," ERL/UCB Memo M79/8, Electronics Research Laboratory, University of California, Berkley, February 1979.

[FT] Fischer, P. and S. Thomas, "Operators for Non-First-Normal-Form Relations," *Proceedings of the 7th International Computer Software Applications Conference*, Chicago (November 1983), 464—475.

[FvG1] Fischer, P. and D. Van Gucht, "Structure of Relations Satisfying Certain Families of Dependencies," Technical Report CS-84-10, Department of Computer Science, Vanderbilt University, August 1984.

[FvG2] Fischer, P. and D. Van Gucht, "Weak Multivalued Dependencies," *Proceedings of the ACM Symposium on Principles of Database Systems*, Waterloo (April 1984), 266–274.

[Fur] Furtado, A., "Horizontal Decomposition to Improve a Non-BCNF Scheme," *ACM SIGMOD Record 12*, 1 (October 1981), 26–32.

[FK] Furtado, A. and L. Kerschberg, "An algebra of quotient relations," *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Toronto (1977), 1–8.

[HY] Hull, R. and C. Yap, "The Format Model: A Theory of Database Organization," *Journal of the ACM 31*, 3 (July 1984), 518–537.

[Jac] Jacobs, B., "On Database Logic," *Journal of the ACM 29*, 2 (April 1982), 310-332.

[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 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.

[Klu] Klug, A., "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions," *Journal of the ACM 29*, 3 (July 1982), 699–717.

[KS] Korth, H. and A. Silberschatz, "A User-Friendly Operating Sytem Interface Based on the Relational Data Model," *Proceedings of the International Symposium on New Directions in Computing*, 1985. (Also, TR-84-12, Department of Computer Science, University of Texas at Austin, April 1984).

[KV] Kuper, G. and M. Vardi, "A New Approach to Database Logic," *Proceedings of the ACM Symposium on Principles of Database Systems*, Waterloo (April 1984), 86–96.

[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.

[Orm] Orman, L., "Semantics of indexed sets," Cornell University BPA Working Paper (1981).

[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.

[RKS] Roth, M., H. Korth and A. Silberschatz, "Null Values in ¬1NF Relational Databases," TR-85-32, Department of Computer Science, University of Texas at Austin (December 1985).

[Sch] Schek, H.-J., "Towards a Basic Relational NF^2 Algebra Processor," *Foundations of Data Organization*, Kyoto, Japan (1985).

[ScS] 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).

[SmS] Smith, J. and D. Smith, "Database abstractions: aggregation and generalization," *ACM Transactions of Database Systems 2*, 2 (June 1977), 105-133.

[TFS] Thomas, S., P. Fischer and L. Saxton, "Interactions between Dependencies and Nested Relational Databases," Technical Report, Department of Computer Science, Vanderbilt University, 1984.

[Ull] Ullman, J., *Principles of Database Systems*, 2nd Edition, Computer Science Press, 1982.