POSSIBILITIES AND LIMITATIONS OF USING FLAT OPERATORS IN NESTED ALGEBRA EXPRESSIONS

Jan Paredaens
Dept of Math and Computer Science
University of Antwerp
B-2610 Antwerpen, Belgium

and

Dirk Van Gucht
Computer Science Dept
Indiana University
Bloomington, IN 47405, USA

1 Introduction

In 1977 Makinouchi [18] proposed to generalize the relational database model by removing the first normal form assumption Jaeschke and Schek [15] introduced a generalization of the ordinary relational model by allowing relations with set-valued attributes and adding two restructuring operators, the nest and the unnest operators, to manipulate such (one-level) nested relations Thomas and Fischer [26] generalized Jaeschke and Schek's model and allowed nested relations of arbitrary (but fixed) depth Roth, Korth and Silberschatz [23] defined a calculus like query language for the nested relational model of Thomas and Fischer Since then numerous SQL-like query languages [17,20,21,22], graphical-oriented query languages [13] and datalog-like languages [3,4,7,16] have been introduced for this model or slight generalizations of it Also, various groups [5,10,11,12,19,25] have started to implement the nested relational database model, some on top of an existing database management system, others from scratch

In this paper we focus on the nested algebra as proposed by Thomas and Fischer One of the key problems we address is the following "What is the power of the nested algebra when we are only interested in its operation on flat relations and when the result is also flat?" In other words "Is it possible to write queries in the nested algebra with flat operands as input and with a flat output which do not have an equivalent in the ordinary (flat) relational algebra?" We show in this paper that the answer to this question is negative Hence the flat algebra is rich enough to extract the same "flat information" from a flat database as the nested algebra does This result has some interesting theoretical as well as practical consequences

- The transitive closure can not be expressed in the nested algebra, because if it could, the transitive closure could be expressed in the flat algebra, contradicting a result of Aho and Ullman [2] It is interesting to observe that if the nest operator is replaced by the powerset operator in the nested algebra, then it is possible, as observed by Beeri [6], Hull [14] and Abiteboul [1], to express the transitive closure Hence the nested algebra with the powerset operator is strictly more powerful than the classical nested algebra studied in this paper
- Several researchers [19,25] have proposed to build database management systems supporting various

nested relational database models Subsequently, others [8,24] viewed such database management systems as attractive tools to implement (flat) relational database management systems, because additional opportunities are offered to optimize (flat) relational queries Our result indicates that this strategy is safe since no expressive power is gained by using the nested algebra

In addition to this result, we have some positive and negative results about how general queries in the nested algebra can be transformed such that maximal use is made of the application of flat relational operators These theoretical results shed some light on the limitations set by trying to implement the nested relational on top of a relational database management system

2 Formalism

In this section we define relation schemes, relation instances, a nested algebra and a nested calculus

2 1. Scheme of a Relation

$\langle$*Attribute*$\rangle \rightarrow \langle$*Identifier*$\rangle$

Such an attribute is called *atomic* and $\langle$*Identifier*$\rangle$ is called the *name* of the attribute

$\langle$*Attribute*$\rangle \rightarrow \langle$*Identifier*$\rangle\langle$*Scheme*$\rangle$

Such an attribute is called *structured* and $\langle$*Identifier*$\rangle$ is called the *name* of the attribute

$\langle$*List_of_attributes*$\rangle \rightarrow \langle$*Empty_list*$\rangle$ |
$\langle$*Non_empty_list_of_attributes*$\rangle$

$\langle$*Non_empty_list_of_attributes*$\rangle \rightarrow \langle$*Attribute*$\rangle$ |
$\langle$*Attribute*$\rangle\underline{,}\langle$*Non_empty_list_of_attributes*$\rangle$

Two lists of attributes λ_1 and λ_2 are called *compatible* if and only if they are both empty, or $\lambda_1 = \alpha_1, \lambda_1'$ and $\lambda_2 = \alpha_2, \lambda_2'$ with λ_1', λ_2' compatible lists of attributes and α_1 and α_2 both atomic attributes or both structured attributes with compatible lists of attributes

$\langle$*Scheme*$\rangle \rightarrow \underline{(}\langle$*Non_empty_list_of_attributes*$\rangle\underline{)}$

All the identifiers in a scheme have to be different Two schemes are called *compatible* if and only if their respective lists of attributes are compatible A scheme is called *flat* if and only if all its attributes are atomic

These are some examples of schemes

(B_1, B_2, B_3)

$(A, B, C(D, E(F)))$

A *database scheme* is a set of schemes

2 2 Instances of a Relation Scheme

Let S be the scheme $(\alpha_1, \quad, \alpha_n)$, where α_i stands for an attribute, either atomic or structured The set of *instances* of S, denoted *Inst*(S), is the set

$\{s | s$ is a finite subset of *values*$(\alpha_1) \times \quad \times$ *values*$(\alpha_n)\}$

where *values*(A) is the set of the natural numbers if A is an atomic attribute and *values*$(A(\lambda)) =$ *Inst*$((\lambda))$ otherwise (where λ is a non-empty list of attributes)

We need to make the following remarks

- All atomic attributes have the same *values* set, namely the set of the natural numbers
- Two schemes are compatible if and only if their set of instances are equal

2.3. Nested Algebra

In this section we define a nested algebra for manipulating schemes and their instances similar to the one introduced by Thomas and Fischer [26] It should be noted that the empty operator and the renaming operator were not considered in [26] The empty operator is introduced to effectively deal with instances containing tuples with empty component values The algebra consists of 9 operators, which are defined as follows

- *The union operator* $\cup$ let (λ_1) and (λ_2) be compatible schemes and let $s_1 \in$ *Inst*$((\lambda_1))$ and $s_2 \in$ *Inst*$((\lambda_2))$ Then $s_1 \cup s_2$ is the (standard) union of s_1 and s_2 and is an instance of the scheme (λ_1)
- *The difference operator* $-$ let (λ_1) and (λ_2) be compatible schemes and let $s_1 \in$ *Inst*$((\lambda_1))$ and $s_2 \in$ *Inst*$((\lambda_2))$ Then $s_1 - s_2$ is the (standard) difference of s_1 and s_2 and is an instance of the scheme (λ_1)
- *The join operator* $\bowtie$ let (λ_1) and (λ_2) be schemes with no common identifiers and let $s_1 \in$ *Inst*$((\lambda_1))$ and $s_2 \in$ *Inst*$((\lambda_2))$ Then $s_1 \bowtie s_2$ is the (standard) join of s_1 and s_2 and is an instance of the scheme (λ_1, λ_2)
- *The projection operator* Π Let (λ) be the scheme $(\alpha_1 \quad, \alpha_k)$, let $i_1, \quad, i_k$ $(k \geq 1)$ be the indices of different attributes $\alpha_{i_1}, \quad, \alpha_{i_k}$ of λ and let $s \in$ *Inst*$((\lambda))$ Then $\Pi_{i_1, \quad, i_k}(s)$ is the (standard) projection on the α_{i_1} through α_{i_k} attributes and is an instance of the scheme $(\alpha_{i_1}, \quad, \alpha_{i_k})$

- *The selection operator σ* Let (λ) be a scheme, let i and j be the indices of different attributes of (λ) and let $s \in Inst((\lambda))$ Then $\sigma_{i=j}(s)$ is the largest subset of s consisting of elements with equal i and j components
- *The nest operator ν* Let (λ) be the scheme (λ_1, λ_2), where λ_2 is not the empty list and let A be no identifier of λ Let $s \in Inst((\lambda))$ Then $\nu_{\lambda_2,A}(s)$ is the (standard) nest of s on the attributes of λ_2 and is an instance of the scheme $(\lambda_1, A(\lambda_2))$ Notice that in this definition we have made the notational simplifying assumption that λ_2 is an end sequence of λ and not a subsequence of λ
- *The unnest operator μ* Let (λ) be the scheme $(\lambda_1, A(\lambda_2))$ and let $s \in Inst((\lambda))$ Then $\mu_{A(\lambda_2)}(s)$ is the (standard) unnest of s on the $A(\lambda_2)$ attribute of λ and is an instance of the scheme (λ_1, λ_2)
- *The empty operator $\emptyset$* Let (λ) be a scheme and let A be no identifier of λ Let $s \in Inst((\lambda))$ Then $\emptyset_A(s)$ is an instance of the scheme $(A(\lambda))$ that has only one tuple, being the empty instance of the scheme (λ)
- *The renaming operator ρ* Let (λ) be a scheme and let $s \in Inst((\lambda))$ Then $\rho(s, A, B)$ is the (standard) renaming of A by B in s It is an instance of the scheme (λ), with the identifier A substituted by the identifier B

Algebraic expressions of the nested algebra are defined in the usual way An expression in the nested algebra is called

- *nested-nested (nn)* if and only if at least one of its operands and its result are not flat
- *flat-nested (fn)* if and only if its operands are flat but its result is not flat
- *nested-flat (nf)* if and only if at least one of its operands is not flat but its result is flat
- *flat-flat (ff)* if and only if its operands and its result are flat

2 4 Nested Calculus

In this section we define a calculus for manipulating schemes and there instances, similar to the one introduced by Roth, Korth and Silberschatz [23]

A *query* in the nested calculus has the form

$$\{t[\lambda] | f(t)\}$$

where t is an identifier, called the ***target variable*** and λ is a non-empty list of attributes, called the ***scheme*** of t and $f(t)$ is a ***formula***, called the formula of the query The scheme of this query is (λ) The free variables of this query are those of f, except for t A formula can have the form

$$(f_1 \vee \quad \vee f_n)$$
$$(f_1 \wedge \quad \wedge f_n)$$
$$\neg f_1$$
$$\exists t[\lambda] f_1$$

where all f_i's are formulas, t is a an identifier, called a tuple variable (all these t's are different, and different from the target variable), and λ is a non-empty list of attributes, called the scheme of t Furthermore, a formula can have the form

$$\langle term_1 \rangle = \langle term_2 \rangle$$
$$\langle term_1 \rangle \in \langle term_2 \rangle$$

A *term* is a query or has the form

t

$t(\alpha)$ ($t(\alpha)$ is called a *component* of t)

R

where t is a tuple variable, R a relation identifier and α is an attribute (atomic or structured) The scheme of $t(A)$, where A is an atomic attribute, is A The scheme of $t(A(\lambda))$, where λ is a non-empty list of attributes and $A(\lambda)$ is a structured attribute, is (λ)

Remark that $\langle term_1 \rangle = \langle term_2 \rangle$ is only a formula if the schemes of $\langle term_1 \rangle$ and $\langle term_2 \rangle$ are compatible and that $\langle term_1 \rangle \in \langle term_2 \rangle$ is only a formula if the scheme of $\langle term_1 \rangle$ is λ and the scheme of $\langle term_2 \rangle$ is compatible with (λ)

Here are some examples of queries, R have the scheme $(A, B(C, D))$

$$\{t[A] | \exists t_1[A, B(C, D)](t_1 \in R \wedge t(A) = t_1(A))\}$$
$$\{t[A, B(C, D)] | \neg t \in R\}$$
$$\{t_1[C] | \exists t_2[B(C)](t_2(B(C)) = \{t_3[C] | t_3 \in t_2(B(C))\} \wedge t_1(C) \in t_2(B(C)))\}$$

In the last query, $t_2(B(C))$ is defined in terms of itself which causes a lot of problems As a matter of fact every set with the good scheme satisfies this query

2 5 Flat Algebra

An expression of the nested algebra is also an expression of the *flat algebra* if and only if

1 every attribute of every relation that occurs in the expression is atomic, and
2 the nest operator nor the empty operator does not occur in the expression

2.6. Flat Calculus

A query of the nested calculus is also a query of the *flat calculus* if and only if

1 every attribute of every relation that occurs in the query is atomic, and
2 every attribute of the scheme of every variable that occurs in the query is atomic, and
3 no term is a query

3. Translation of the Nested Algebra to the Nested Calculus

In this section we show how expressions of the nested algebra can be translated into the nested calculus

Let nae, nae_1, and nae_2 be expressions of the nested algebra

- R, a relation with scheme (λ) is translated in $\{t[\lambda] | t \in R\}$

We suppose that the nested algebraic expressions nae, nae_1 and nae_2 have as translation the queries $\{t[\lambda] | f(t)\}$, $\{t[\lambda_1] | f_1(t)\}$ and $\{t[\lambda_2] | f_2(t)\}$ respectively

- *Union* $nae_1 \cup nae_2$ (where λ_1 and λ_2 are compatible) is translated to $\{t[\lambda_1] | (f_1(t) \vee f_2(t))\}$
- *Difference* $nae_1 - nae_2$ (where λ_1 and λ_2 are compatible) is translated to $\{t[\lambda_1] | (f_1(t) \wedge \neg f_2(t))\}$
- *Join* $nae_1 \bowtie nae_2$ (where λ_1 and λ_2 have no common identifiers) is translated to

$$\{t[\lambda_1, \lambda_2] | (\exists t_1[\lambda_1](f_1(t_1) \bigwedge_{\alpha \in \lambda_1} t(\alpha) = t_1(\alpha)) \wedge$$
$$\exists t_2[\lambda_2](f_2(t_2) \bigwedge_{\alpha \in \lambda_2} t(\alpha) = t_2(\alpha)))\}$$

- *Projection* $\Pi_{i_1 \quad i_k}(nae)$ (where $k \geq 1$ and $i_1, \quad , i_k$ are the indices of different attributes $\alpha_{i_1}, \quad , \alpha_{i_k}$ of λ) is translated to

$$\{t[\alpha_{i_1}, \quad , \alpha_{i_k}] | \exists t_1(\lambda)(f(t_1) \bigwedge_{j=1}^{k} t(\alpha_{i_j}) = t_1(\alpha_{i_j}))\}$$

- *Selection* $\sigma_{i=j}(nae)$ (where i and j are the indices of different attributes α_i and α_j of λ) is translated to $\{t[\lambda] | (f(t) \wedge t(\alpha_i) = t(\alpha_j))\}$

- *Nest* suppose that λ can be written as λ^1, λ^2 where λ^2 is a non-empty list of attributes and let A be no identifier of λ $\nu_{\lambda^2, A}(nae)$ is translated to

$$\{t[\lambda^1, A(\lambda^2)] | \exists t_1[\lambda](f(t_1) \bigwedge_{\alpha \in \lambda^1} t(\alpha) = t_1(\alpha) \wedge$$
$$t(A(\lambda^2)) = \{t_2[\lambda^2] | \exists t_3[\lambda](f(t_3)$$
$$\bigwedge_{\alpha \in \lambda^1} t(\alpha) = t_3(\alpha) \bigwedge_{\alpha \in \lambda^2} t_2(\alpha) = t_3(\alpha))\})\}$$

- *Unnest* let $\lambda = \lambda^1, A(\lambda^2)$ $\mu_{A(\lambda^2)}(nae)$ is translated to

$$\{t[\lambda^1, \lambda^2] | \exists t_1[\lambda] \exists t_2[\lambda^2](f(t_1) \bigwedge_{\alpha \in \lambda^1} t(\alpha) = t_1(\alpha) \wedge$$
$$t_2 \in t_1(A(\lambda^2)) \bigwedge_{\alpha \in \lambda^2} t(\alpha) = t_2(\alpha))\}$$

- *Empty operator* let (λ) be the scheme of nae and let A be no identifier of λ $\emptyset_A(nae)$ is translated to

$$\{t[A(\lambda)] | t(A(\lambda)) = \{t_1[\lambda] | f(t_1) \wedge \neg f(t_1)\}\}$$

- *Renaming* $\rho(nae, A, B)$ is translated to the translation of nae where every occurrence of A is substituted by B

4 Translation of ff-Expressions into Expressions of the Flat Algebra

In this section we show that ff-expressions of the nested algebra can be translated into an equivalent expression of the flat algebra To establish this result we first translate the ff-expression into a calculus query which satisfies desirable properties We call such a query constructive We then translate, with the help of several transformation rules, the constructive query into a safe query (in the sense of Ullman [27]) of the flat calculus We finally use Codd's theorem [9,27] about the equivalence of the safe flat calculus and the flat algebra We would like to state that it is an open problem to find a completely algebraic strategy to achieve this result

4.1. Existential Normal Form

A formula g is in *existential normal form (enf)* if and only if

$$g \equiv (h_1 \vee \quad \vee h_k) \qquad k \geq 1$$

such that for all i, $1 \leq i \leq k$

$$h_i \equiv \exists v_{i1}[\lambda_{i1}] \quad \exists v_{in_i}[\lambda_{in_i}](c_{i1} \wedge \quad \wedge c_{il_i} \wedge \neg d_i) l_i \geq 0$$

such that d_i is in enf and

$$c_{ij} \equiv t_1(\alpha_1) = t_2(\alpha_2), \text{ or}$$
$$c_{ij} \equiv t_1 \in t_2(\alpha), \text{ or}$$
$$c_{ij} \equiv t_1(A(\lambda)) = \{t_2[\lambda] \| f\}, \text{ where } f \text{ is in enf, or}$$
$$c_{ij} \equiv \{t_2[\lambda] \| f\} = t_1(A(\lambda)), \text{ where } f \text{ is in enf, or}$$
$$c_{ij} \equiv t \in R, \text{ or}$$
$$c_{ij} \equiv t_1 \in \{t_2[\lambda] \| f\}, \text{ where } f \text{ is in enf, or}$$
$$c_{ij} \equiv \{t_1[\lambda_1] \| f_1\} = \{t_2[\lambda_2] \| f_2\}, \text{ where } f_1 \text{ and } f_2 \text{ are in enf}$$

The formulas of the three examples in Section 2 4 are in enf

4.2 Constructibility Graph, Reachability, Acyclicity and Constructiveness

Let g be in enf and let $F = \{u_1(\alpha_{p_1}), \quad , u_l(\alpha p_l)\}$ be a set of components of the variables $u_1, \quad , u_l$ (these are called the variables of F) with

$$g \equiv (h_1 \vee \quad \vee h_k)$$
$$h_i \equiv \exists v_{i1}[\lambda_{i1}] \quad \exists v_{in_i}[\lambda_{in_i}](c_{i1} \wedge \quad \wedge c_{il_i} \wedge \neg d_i)$$

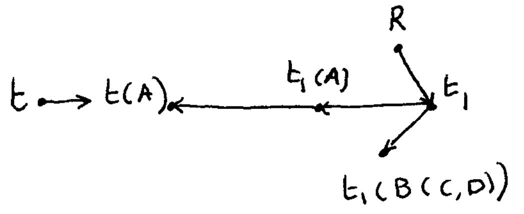
The *constructibility graph* of (h_i, F) is the directed graph, notated by $C(h_i, F)$ with nodes

1 all components of the variables $v_{i1}, \quad , v_{in_i}$ and all the elements of F
2 all the variables $v_{i1}, \quad , v_{in_i}$ and the variables $u_1, \quad , u_l$
3 all queries Q that appear in h_i on the highest level in some c_{ij}
4 all relations R that appear in h_i on the highest level in some c_{ij}

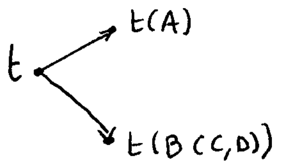
and with edges

1 if $x(\alpha) = y(\beta)$ is one of the c_{ij}'s then $(x(\alpha), y(\beta))$ is an edge if $x(\alpha) \notin F$ and $(y(\beta), x(\alpha))$ is an edge if $y(\beta) \notin F$
2 if $x \in R$ is one of the c_{ij}'s then (R, x) is an edge
3 if $x \in y(\alpha)$ is one of the c_{ij}'s then $(y(\alpha), x)$ is an edge if $y(\beta) \notin F$
4 if $x \in Q$ is one of the c_{ij}'s, Q being a query, then (Q, x) is an edge
5 if $x(\alpha) = Q$ or $Q = x(\alpha)$ is one of the c_{ij}'s, Q being a query, then $(Q, x(\alpha))$ is an edge
6 if x is a variable and $x(\alpha)$ is a component of x then $(x, x(\alpha))$ is an edge

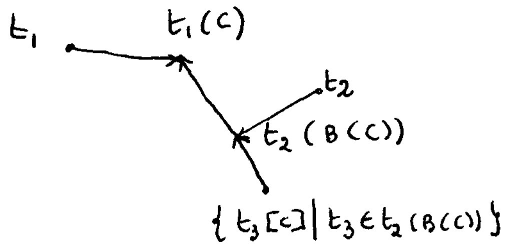
The constructibility graphs of the three examples in Section 2 4 are shown in Figure 1



$$C(\exists t_1[A, B(C,D)](t_1 \in R \wedge t(A) = t_1(A)), \{t(A)\})$$



$$C(\neg t \in R, \{t(A), t(B(C,D))\})$$



$$C(\exists t_2[B(C)](t_2(B(C)) = \{t_3[C] \| t_3 \in t_2(B(C))\} \wedge t_1(C) \in t_2(B(C))), \{t_1(C)\})$$

Figure 1 The constructibility graphs of the three examples in Section 2 4

A component $x(\alpha)$ is called *reachable* in a subgraph $C'(h_i, F)$ of $C(h_i, F)$ if and only if there is a path in $C'(h_i, F)$ from a query Q or a relation R to $x(\alpha)$ A component $x(\alpha)$ is called *reachable* in (h_i, F) if and only if $x(\alpha)$ is reachable in $C(h_i, F)$ The component

$x(\alpha)$ is called *reachable* in (g, F) if and only if it is reachable in (h_i, F), for all i, $1 \leq i \leq k$.

(h_i, F) is called *acyclic* if and only if $C(h_i, F)$ has a subgraph $A(h_i, F)$ such that

1 all the components of the tuple variables v_{i1}, , v_{in_i} and all the elements of F are reachable in $A(h_i, F)$, and
2 $A(h_i, F)$ augmented with all the edges $(x(\alpha), Q)$, with $x(\alpha)$ appearing in the query Q, is acyclic We will call the latter graph $AU(h_i, F)$

(h_i, F) is called *constructive* if and only if

1 (h_i, F) is acyclic, and
2 $(d_i, \emptyset)$ is constructive, and
3 if some c_{ij} has the form

$$v(\alpha) = \{w[\gamma] \| f\}, \text{ or}$$
$$\{w[\gamma] \| f\} = v(\alpha), \text{ or}$$
$$v \in \{w[\gamma] \| f\}$$

then $\{w[\gamma] \| f\}$ is constructive

4 if some c_{ij} has the form

$$\{w_1[\gamma_1] \| f_1\} = \{w_2[\gamma_2] \| f_2\}$$

then $\{w_1[\gamma_1] \| f_1\}$ and $\{w_2[\gamma_2] \| f_2\}$ are constructive

(g, F) is called *constructive* if and only if for all i, $1 \leq i \leq k$, (h_i, F) is constructive The query $\{u[\lambda] \| g\}$ is called *constructive* if and only if (g, F) is constructive, where $F = \{u(\alpha) | \alpha \in \lambda\}$ Note that, whenever $G \subset F$, (g, G) is constructive if (g, F) is also constructive

The first example of Section 2 4 is constructive, the second is not (since no component of t is reachable in $(\neg t \in R, \{t(A), t(B(C))\})$), nor is the the third (since the graph AU does not exist)

In the following lemma, we present some equivalences between logical formulas which are used in the proof of Lemma 2 and Lemma 3

Lemma 1 For each formula f_1 and f_2 we have

E1 If t_2 is not a free variable of f_1 then

$$(f_1 \wedge \exists t_2[\lambda] f_2(t_2))$$

is logically equivalent to

$$\exists t_2[\lambda](f_1 \wedge f_2(t_2))$$

E2 If t_2 is not a free variable of $f_1(t_1)$ and t_1 is not a free variable of $f_2(t_2)$ then

$$(\exists t_1[\lambda_1] f_1(t_1) \wedge \exists t_2[\lambda_2] f_2(t_2))$$

is logically equivalent to

$$\exists t_1[\lambda_1] \exists t_2[\lambda_2](f_1(t_1) \wedge f_2(t_2))$$

E3

$$\exists t[\lambda](f_1(t) \vee f_2(t))$$

is logically equivalent to

$$(\exists t[\lambda] f_1(t) \vee \exists t[\lambda] f_2(t))$$

In Lemma 2 and Lemma 3 we show how constructiveness is preserved in various situations

Lemma 2 Let f_1 and f_2 be formulas and let F and G be sets of components

A If (f_1, F) and (f_2, F) are constructive, then there exists a formula, denoted $f_1 \underline{\vee} f_2$, which is logically equivalent to $f_1 \vee f_2$ and such that $(f_1 \underline{\vee} f_2, F)$ is constructive

B If (f_1, F) and (f_2, G) are constructive and and the components of the common variables of f_1 and f_2 belong to $F \cap G$ then there exists a formula, denoted $f_1 \underline{\wedge} f_2$, which is logically equivalent to $f_1 \wedge f_2$ and such that $(f_1 \underline{\wedge} f_2, F \cup G)$ is constructive

C If (f_1, F) and $(f_2, \emptyset)$ are constructive, then there exists a formula, denoted $f_1 \underline{\wedge \neg} f_2$ which is logically equivalent to $f_1 \wedge \neg f_2$ and such that $(f_1 \underline{\wedge \neg} f_2, F)$ is constructive

D If (f_1, F) is constructive and F contains the set of all the components of the variable t_1, then there exists a formula, denoted $\underline{\exists} t_1[\lambda_1] f_1$ which is logically equivalent to $\exists t_1[\lambda_1] f_1$ and such that $(\underline{\exists} t_1[\lambda_1] f_1, F')$ is constructive, F' being the set of components of F that are not components of t_1

Lemma 3 Let $\underline{\exists} t_1[\lambda_1] f_1$ be a formula and let F be a set of components

A If $(\underline{\exists} t_1[\lambda_1] f_1, F)$ is constructive and $t(\beta)$ does not appear in $\underline{\exists} t_1[\lambda_1] f_1$, then there exists a formula, denoted $\underline{\exists} t_1[\lambda_1](f_1 \underline{\wedge} t_1(\alpha) = t(\beta))$, which is logically equivalent to $\underline{\exists} t_1[\lambda_1](f_1 \wedge t_1(\alpha) = t(\beta))$ and such that $(\underline{\exists}(t_1[\lambda_1] f_1 \underline{\wedge} t_1(\alpha) = t(\beta), F \cup \{t(\beta)\})$ is constructive

B If (f, F) is constructive and $u(\beta_1)$ and $u(\beta_2)$ are components in F, then there exists a formula, denoted $f \Delta u(\beta_1) = u(\beta_2)$, which is logically equivalent to $f \wedge u(\beta_1) = u(\beta_2)$ and such that $(f \Delta u(\beta_1) = u(\beta_2), F)$ is constructive

C If $(\underline{\exists} t_1[\lambda_1] f_1, F)$ is constructive, $t_1(A(\lambda))$ is a component of t_1 and no component of t nor t itself appears in $\underline{\exists} t_1[\lambda_1] f_1$, then there exists a formula, denoted $\underline{\exists} t_1[\lambda_1](f_1 \Delta t \in t_1(A(\lambda)))$, which is logically equivalent to $\underline{\exists} t_1[\lambda_1](f_1 \wedge t \in t_1(A(\lambda)))$ and such that $(\underline{\exists} t_1[\lambda_1](f_1 \Delta t \in t_1(A(\lambda))), F \cup \{t(\beta) | \beta \in \lambda\})$ is constructive

D If $(\underline{\exists} t_1[\lambda_1] f_1, F)$ is constructive, Q is a constructive query and $t(\alpha)$ does not appear in $\underline{\exists} t_1[\lambda_1] f_1$, then there exists a formula, denoted $\underline{\exists} t_1[\lambda_1](f_1 \Delta\ t(\alpha) = Q)$ which is logically equivalent to $\underline{\exists} t_1[\lambda_1](f_1 \wedge t(\alpha) = Q)$ and such that $(\underline{\exists} t_1[\lambda_1] f_1 \Delta\ t(\alpha) = Q, F \cup \{t(\alpha)\})$ is constructive

Lemma 2 and Lemma 3, together with Section 3 where we talked about translating a nested algebra expression into a nested calculus query, allow us to establish the following result

<u>Lemma 4</u> Every expression of the nested algebra can be translated into a constructive query

4.3. Transformations Between Logically Equivalent Formulas

We now define four query transformation rules that preserve constructiveness and that will allow for the "flattening" of constructive queries

Let g be a formula and let F be a set of components of a tuple variable and let (g, F) be constructive, with

$$g \equiv h_1 \vee \quad \vee h_{i-1} \vee h_i \vee h_{i+1} \vee \quad \vee h_k$$
$$h_i \equiv \exists v_1[\lambda_1] \quad \exists v_j[\lambda_j] \quad \exists v_n[\lambda_n] \quad (c_1 \wedge \quad \wedge c_l \wedge \quad \wedge c_m \wedge \neg d)$$

<u>T1 *Query Membership Elimination*</u>

Let $c_l \equiv t_1 \in \{t_2[\lambda_2] || f(t_2)\}$ The query membership elimination substitutes g by g' with

$$g' \equiv h_1 \vee \quad \vee h_{i-1} \vee h'_i \vee h_{i+1} \vee \quad \vee h_k$$
$$h'_i \equiv \exists v_1[\lambda_1] \quad \exists v_n[\lambda_n] \quad ((c_1 \wedge \quad \wedge c_{l-1} \wedge c_{l+1} \wedge \quad \wedge c_m \wedge \neg d) \Delta f(t_1))$$

<u>T2 *Query Propagation*</u>

Let $c_l \equiv t_1(\alpha) = Q$ or $c_l \equiv Q = t_1(\alpha)$ where $Q \equiv \{t_2[\lambda_2] || f(t_2)\}$ and let the edge $(Q, t_1(\alpha))$ belong to $AU(h_i, F)$ The query propagation substitutes g by g' with

$$g' \equiv h_1 \vee \quad \vee h_{i-1} \vee h'_i \vee h_{i+1} \vee \quad \vee h_k$$
$$h'_i \equiv \exists v_1[\lambda_1] \quad \exists v_n[\lambda_n] \quad (c'_1 \wedge \quad \wedge c'_{l-1} \wedge c_l \wedge c'_{l+1} \wedge \quad \wedge c'_m \neg d')$$

where c'_p $(p \neq l)$ and d' is c_p and d respectively with every occurrence of $t_1(\alpha)$ replaced by Q

<u>T3 *Structured Component Elimination*</u>

Consider the formula h_i and suppose that c_l is the only term in which $v_j(A(\lambda))$ occurs The structured component elimination consists of

1 Deletion of the term c_l from h_i
2 Delete the attribute $A(\lambda)$ from the scheme of v_j
3 If the scheme of v_j becomes empty, remove $\exists v_j[\,]$

Call the resulting formula h'_i and let

$$g' \equiv h_1 \vee \quad \vee h_{i-1} \vee h'_i \vee h_{i+1} \vee \quad \vee h_k$$

<u>T4 *Query Equality Elimination*</u>

Let $c_l \equiv \{t_1[\gamma_1] || f_1(t_1)\} = \{t_2[\gamma_2] || f_2(t_2)\}$ The query equality elimination substitutes g by g' with

$$g' \equiv h_1 \vee \quad \vee h_{i-1} \vee h'_i \vee h_{i+1} \vee \quad \vee h_k$$
$$h'_i \equiv \exists v_1[\lambda_1] \quad \exists v_n[\lambda_n](c_1 \wedge \quad \wedge c_{l-1} \wedge c_{l+1} \wedge \quad \wedge c_m \wedge \neg(d \underline{\vee} \underline{\exists} t_1[\gamma_1](f_1(t_1) \underline{\wedge \neg} f_2(t_1)) \underline{\vee} \underline{\exists} t_2[\gamma_2](f_2(t_2) \underline{\wedge \neg} f_1(t_2))))$$

<u>Lemma 5</u> The query membership elimination rule, the query propagation rule, the structured component elimination rule and the query equality elimination rule preserve constructiveness

<u>Lemma 6</u> A constructive query in the flat calculus is safe

4.4. Algorithm Translate

We now present an algorithm to construct a query in the flat calculus that is logically equivalent to a formula of the nested algebra with flat operands and a flat result

Translate

Input a ff-expression of the nested algebra

Output an expression of the flat algebra

Method

1 Transform the given expression of the nested algebra into a constructive query, using Lemma 4
2 Repeat step 2 1 and step 2 2 until no more possible
   2 1 Repeat query membership elimination until no more possible
   2 2 Repeat query propagation immediately followed by structured component elimination until no more possible
3 Repeat query equality elimination until no more possible
4 Apply Codd's theorem about translating a safe calculus query into an equivalent flat algebra expression

Lemma 7 Algorithm *Translate*, taking as input a ff-expression of the nested algebra, stops and produces an expression of the flat algebra which is equivalent with the given ff-expression

We are now able to formulate one of our main results

Theorem 1 For every ff-expression in the nested algebra there is an equivalent expression in the flat algebra

Corollary 1 The transitive closure is not expressible in the nested algebra

5. Normalizing nn-Expressions, fn-Expressions and nf-Expressions

The nested algebra is richer than the flat algebra, since it can extract "nested information" from a nested database, which cannot be expressed by the flat algebra This information cannot be represented by a flat relation, not even intermediately This is shown in the next theorem

Theorem 2 Let *nae* be an arbitrary expression in the nested algebra There are no nf-expression Φ_{nae}, no fn-expression Ψ_{nae} and no expression in the flat algebra e_{nae} with $nae = \Psi_{nae}\ e_{nae}\ \Phi_{nae}$

We are now going to discuss the nested information that can be extracted from a flat database We show that the flat algebra is nearly as powerful as the nested algebra in this context

We first need to generalize the nest operator There fore, consider two relation schemes (λ_1) and (λ_1, λ_2) Consider the operator $g\nu_A$ that associates with these two schemes the scheme $(\lambda_1, A(\lambda_2))$ Suppose that A is the name of no attribute of λ_1 nor of λ_2 $g\nu_A$ associates with the two instances r_1 and r_2 (of respectively (λ_1) and (λ_1, λ_2)) the instance of $(\lambda_1, A(\lambda_2))$ of the tuples, the λ_1-projection of which is in r_1 or in the λ_1-projection of r_2, and the A-component contains all the λ_2-*projections of the tuples in* r_2 *that have the associated* λ_1-projection This operator is called the *general nest*

Consider Figure 2 Clearly $g\nu_A(r_1, r_2) = r_3$

$r_1 =$

C
1
2

$r_2 =$

C	B
0	0
0	1
1	1

$r_3 =$

C	A(B)
0	{0,1}
1	{1}
2	{}

Figure 2

The next two lemmas prove that we can substitute the nest operator by the general nest operator, without changing the expressive power of the nested algebra

Lemma 8 The general nest operator is expressible in the nested algebra

Lemma 9 The nest operator can be expressed in terms of general nest and the rest of the nested algebra

The next theorem essentially states that the calculation of every fn-expression can be simulated by first calculating in the flat algebra, and then nesting in an appropriate way in order to get the answer In general this general nesting is interweaved with projections and renamings

Theorem 3 Let *nae* be an fn-expression in the nested algebra There is an fn-expression Ψ_{nae}, consisting only of renamings, projections and general nests, and $e_{nae^1}, \quad , e_{nae^k}$ expressions in th flat algebra with

$$nae = \Psi_{nae}(e_{nae})(e_{nae^1}, \quad , e_{nae^k})$$

Finally we discuss the flat information extracted from a nested database For this purpose the flat algebra is essentially less powerful than the nested algebra We generalize the unnest operator as follows The *general unnest operator* on the structured attribute A associates with the scheme $(\lambda_1, A(\lambda_2))$ the scheme $(\lambda_1, \lambda_2, A'(\lambda_2'))$ It associates with the instance r the unnest of r, with each tuple augmented by its original A-value (being primed) The general unnest of r on A is denoted by $g\mu_{A(\lambda_2)}(r)$

Consider Figure 3 It illustrates an instance r with its general unnest $g_{\mu_{A(B)}}(r)$

$r =$

C	$A(B)$
0	{0,1}
1	{1}
1	{2}
2	{}

$g_{\mu_{A(B)}}(r) =$

C	B	$A'(B')$
0	0	{0,1}
0	1	{0,1}
1	1	{1}
1	2	{2}

Figure 3

The next two lemmas prove that we can substitute the unnest operator by the general unnest operator without changing the expressive power of the nested algebra

Lemma 10 The general unnest operator can be expressed in the nested algebra

Lemma 11 The unnest operator can be expressed in terms of general unnest and the rest of the nested algebra

Theorem 4 Let *nae* be an nf-expression in the nested algebra There are nf-expressions Ψ_{nae^j}, consisting of no union, empty operator or general nest and e_{nae} an expression in the flat algebra with

$$nae = e_{nae}(\Psi_{ne^1}, \quad , \Psi_{ne^k})$$

The theorem above states that we can express an nf-expression without using the general nest and by postponing the union, such that it is only applied on flat relations We show now that in general we cannot postpone the selection, the join, the difference, the renaming and the projection until their flat application This proves that Theorem 4 cannot be strengthened

Theorem 5 Consider an arbitrary nf-expression nae_1 In general there is no nf-expression nae_2 without the selection (join, difference, renaming, projection respectively) on nested relations and that is equivalent to nae_1

We proved that starting from flat relations and building nested relations we still have enough information to translate the selection, the join, the difference and the projection on those nested relations into the flat algebra This does not hold if we start with the nested relations (as is proved in the theorem)

References

[1] S Abiteboul, *personal communications*

[2] A V Aho, J D Ullman,"Universality of Data Retrieval Languages", *Proc 6th POPL*, San-Antonio, Texas, Jan 1979, pp 110–117

[3] F Bancilhon, "A Logic Programming/Object Oriented Cocktail", *SIGMOD Rec 15 3*, Sept 1986, pp 11–20

[4] F Bancilhon, S Khoshafian, "A Calculus for Complex Objects", *Proc 5th PODS*, Cambridge, Mass , 1986, pp 53–59

[5] F Bancilhon, P Richard, M Scholl, "On Line Processing of Compacted Relations", *Proc 8th VLDB*, Mexico City, 1982, pp 263–269

[6] C. Beeri, *personal communications.*

[7] C. Beeri, S Naqvi, R Ramakrishnan, O Shmueli, S Tsur, "Sets and Negation in a Logic Database Language (LDL1)", *Proc 6th PODS*, San Diego, 1987, pp. 21–37

[8] N Bidoit, "Efficient Evaluation of Relational Queries Using Nested Relations", *Rapports de Recherche, no 480*, INRIA, 1986

[9] E F Codd, "Relational Completeness of Database Sublanguages", *Database Systems*, R Rustin eds , Prentice-Hall, Englewood Cliffs, 1972, pp 65–98

[10] P Dadam, K Kuespert, F Andersen, H Blanken, R Erbe, J Guenauer, V Lum, P Pistor, G Walch, "A DBMS Prototype to Support Extended NF2 Relations An Integrated View on Flat Tables and Hierarchies", *Proc Ann SIGMOD Conf*, Austin, 1986, pp 356–366

[11] U Deppisch, H -B Paul, H - J Schek, "A Storage System for Complex Objects", *Proc Int Workshop on Object-Oriented Database Systems*, Pacific Grove, 1986, pp 183–195

[12] A Deshpande, D Van Gucht, "A Storage Structure for Unnormalized Relations", *Proc GI Conf on Database Systems for Office Automation, Engineering and Scientific Applications*, Darmstadt, April 1987, pp 481–486

[13] G Houben, J Paredaens, "The R^2-Algebra An Extension of an Algebra for Nested Relations", *Tech Rep*, Tech University, Eindhoven, 1987

[14] R Hull, *personal communications*

[15] G Jaeschke, H -J Schek, "Remarks on the the Algebra on Non First Normal Form Relations", *Proc 1st PODS*, Los Angeles, 1982, pp 124–138

[16] G M Kuper, "Logic Programming With Sets", *Proc 6th PODS*, San Diego, 1987, pp 11–20

[17] V Linnemann, "Non First Normal Form Relations and Recursive Queries An SQL-Based Approach", *Proc 3rd IEEE Int Conf on Data Engineering*, Los Angeles, 1987

[18] A Makinouchi, "A Consideration of Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model", *Proc 3rd VLDB*, Tokyo, 1977, pp 447–453

[19] H -B Paul, H -J Schek, M H Scholl, G Weikum, U Deppisch, "Architecture and Implementation of the Darmstadt Database Kernel System", *Proc Ann SIGMOD Conf*, San Francisco, 1987, pp 196–207

[20] P Pistor, F Andersen, "Designing a Generalized NF^2 Model with an SQL-Type Language Interface", *Proc 12th VLDB*, Kyoto, 1986, pp 278–288

[21] P Pistor, R Traunmueller, "A Database Language for Sets, Lists and Tables", *Information Systems 11 4*, 1986, pp 323–336

[22] M A Roth, H F Korth, D S Batory, "SQL/NF A Query Language for ¬1NF Relational Databases", *Inform Systems 12 1*, 1987, pp 99–114

[23] M A Roth, H F Korth, A Silberschatz, "Theory of Non-First-Normal-Form Relational Databases", *Tech Rep TR-84-36 (Revised January 1986)*, University of Texas, Austin, 1984

[24] M H Scholl, "Theoretical Foundation of Algebraic Optimization Utilizing Unnormalized Relations", *Proc 1st ICDT*, Rome, Italy, Sept 1986, in *Lecture Notes in Computer Science*, 243, G Ausiello and P Atzeni, eds , Springer Verlag, pp 380–396

[25] M H Scholl, H -B Paul, H -J Schek "Supporting Flat Relations by a Nested Relational Kernel" *Proc 13th VLDB*, London, 1987

[26] S J Thomas, P C Fischer, "Nested Relational Structures", *Advances in Computing Research III, The Theory of Databases*, P C Kanellakis, ed , JAI Press, 1986, pp 269–307

[27] J D Ullman, *Principles of Database Systems*, 2nd edition, Computer Science Press, 1982