A Recursive Algebra for Nested Relations

by

Latha S. Colby

Department of Computer Science
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT N0. 259

A Recursive Algebra for Nested Relations

By
Latha S. Colby

August 1988

A Recursive Algebra for Nested Relations

Latha S. Colby
Department of Computer Science
Indiana Unversity, Bloomington, IN

Abstract

Nested relations provide a better representational model for complex objects than ordinary (flat) relations. Most query languages for nested relations, however, reformat the structure of complex objects while accessing and manipulating them. If queries involve attributes of objects that are nested deep inside the structure of the relation scheme, the relation has to be flattened before these queries can be performed. A recursive algebra for nested relations that allows manipulation of complex objects without always having to flatten them is presented in this paper. Queries written in this algebra are more succinct and hence easier to formulate.

1. Introduction

The traditional relational model requires that all values in a relation be atomic, *i.e.*, non-decomposable. Although this model is sufficient for representing objects that have simple domains, complex objects cannot be represented easily. A number of database applications, particularly in the areas of CAD/CAM, office automation, textual data and engineering designs, involve complex objects and the relational model is unsuitable for such applications. Normalization in the relational model causes a lot of fragmentation in the representation of objects. Information about objects and their relationships is scattered over several different flat tables. This in turn causes queries to be slow and complicated since excessive joins have to performed among the various relations in the database.

The nested relational model, also sometimes called NF^2 (non- first normal form), is a generalization of the traditional relational model without the first normal form assumption. This means that attributes of a relation can have non-atomic values. They can have relations as values which can hence be viewed as subrelations of the relation. The nested

relational model allows users to view the database in a way that is closer to their concept of the real world since complex objects can be represented as a whole in a single relation instead of being distributed over several different relations.

Many algebra and calculus based query languages have been proposed for nested relations. In most cases, however, these languages are merely simple extensions of the query languages that were developed for the relational model. In algebra based languages, for example, operators like select and project have similar definitons for both the relational and the nested relational model, *i.e.*, they can only operate at the outermost level of a relation even if it is a nested relation with relations embedded at several different levels. If a query on a nested relation involves attributes that are nested deep inside the structure of the relation, the relation has to be "flattened" until the attributes of interest are at the outermost level. Operations like selection and projection are now performed and the result, in some cases, is transformed back to the structure of the original nested relation. The transformation to and from a nested relation is done in the nested relational algebra using "nest" and "unnest" operators. Hence, although the nested relational model provides a better way of modeling complex objects, most query languages for this model cause the structures representing these objects to be restructured while they are being queried thus defeating the main objective of letting the user operate in an environment that models the real world as closely as possible. Since values of attributes of relations can themselves be relations, nested application of the algebraic operators on these subrelations should be allowed so that a nested relation can be accessed and manipulated at all levels without always having to be restructured. A query language which enables recursive application of the algebraic operators from the topmost level down through the relations at different levels would give us a more natural way of querying the database. The algebra for nested relations that is introduced in this paper has operators that are recursively defined and can hence access and manipulate nested relations at all levels without having to flatten them. In addition, queries written in this algebra are more succinct and hence easier to formulate.

The next section provides a background for the remainder of the paper with a brief set of definitions for the nested algebra. Section 3 introduces the new recursive algebra for nested relations. Section 4 establishes the equivalence of the two algebras and finally, Section 5 gives a summary of the paper.

2. The Nested Relational Model

2.1 Background

In 1970 Codd [3] introduced the relational model for databases which is based on the first normal form (1NF) assumption. A relation is in 1NF if all the attributes in the relation have only atomic (non-decomposable) values. This assumption makes it difficult to model complex objects in certain applications. Makinouchi [11] suggested that the 1NF assumption be relaxed so that attributes can be set-valued. Jaeschke and Schek [9] proposed a generalization of the relational model by allowing relations to have non-atomic or set-valued attributes. Thomas and Fischer [20] generalized the model of Jaeschke and Schek and since then a number of researchers [22,18,1,17,14,15,16,12,23,8] have extended the relational database theory to nested relations. Several groups [2,4,5,7,13,19] are attempting to implement the nested relational model, either directly or on top of an existing DBMS.

The tables in Figure 1 are an example of a database for a stock-brokerage firm represented in the (flat) relational model. Figure 2 shows the same information in a nested relational schema. In Figure 2, the attribute INVESTMENTS is a relation-valued attribute of the relation CLIENTS which in turn has SHARES as a relation-valued attribute. In the example given here there are only two levels of nesting. In general, relations can be nested to any arbitrary but finite depth, *i.e.*, relations can have relation-valued attributes, which can have relation-valued attributes and so on.

CLIENTS

NAME	COMPANY	PURCHASE PRICE	DATE	NO.
John Smith	XEROX	64.50	02/10/83	100
John Smith	XEROX	92.50	08/10/87	500
John Smith	IBM	89.75	06/20/83	200
John Smith	IBM	96.50	11/10/84	100
Jill Brody	EXXON	35.00	01/30/81	100
Jill Brody	EXXON	64.50	01/30/82	100
Jill Brody	EXXON	59.50	02/10/83	200
Jill Brody	FORD	35.50	02/10/83	200
Jill Brody	SEARS	35.75	12/25/87	100

CLIENT INFO.

NAME	ADDRESS
John Smith	311 East 2nd. St. Bloomington, IN 47401
Jill Brody	41 North Main St. Oberlin, OH 44074

STOCK DATA

COMPANY	CURRENT PRICE	LAST DIVIDEND
XEROX	52.25	0.44
IBM	97.50	1.25
EXXON	90.00	0.82
FORD	41.75	0.20
SEARS	77.50	0.34

EXCHANGE DATA

COMPANY	EXCHANGES
XEROX	NEW YORK
IBM	NEW YORK
IBM	LONDON
IBM	HONG KONG
IBM	TOKYO
EXXON	NEW YORK
EXXON	LONDON
EXXON	TOKYO
FORD	NEW YORK
SEARS	NEW YORK

Fig. 1 An Example of a Flat Relational Model

CLIENTS

NAME	ADDRESS	INVESTMENTS			
		COMPANY	SHARES		
			PURCHASE PRICE	DATE	NO.
John Smith	311 East 2nd. St. Bloomington, IN 47401	XEROX	64.50	02/10/83	100
			92.50	08/10/87	500
		IBM	89.75	06/20/83	200
			96.50	11/10/84	100
Jill Brody	41 North Main St. Oberlin, OH 44074	EXXON	35.00	01/30/81	100
			64.50	01/30/82	100
			59.50	02/10/83	200
		FORD	35.50	02/10/83	200
		SEARS	35.75	12/25/87	100

STOCK DATA

COMPANY	CURRENT PRICE	EXCHANGES TRADED	LAST DIVIDEND
		EXCHANGES	
XEROX	52.25	NEW YORK	0.44
IBM	97.50	NEW YORK LONDON HONG KONG TOKYO	1.25
EXXON	90.00	NEW YORK LONDON TOKYO	0.82
FORD	41.75	NEW YORK	0.20
SEARS	77.50	NEW YORK	0.34

Fig. 2 An Example of a Nested Relational Model

2.2 Definitions for the Nested Relational Model

Let $\mathcal{A}$ be the universal set of attribute names and relation scheme names. A relation scheme of a relation is of the form $R(S)$ where $R \in \mathcal{A}$ is the relation scheme name and S is a list of the form $(A_1, A_2, ..., A_n)$ where each A_i is either an atomic attribute or a relation scheme of a sub-relation. If A_i is a relation scheme of the form $R_i(S_i)$, then R_i, the name of the scheme, is called a relation-valued attribute of R.

Let $\mathcal{D}$ be the domain of all the atomic attributes in $\mathcal{A}$. An instance r of a relation scheme $R(S)$, where $S = (A_1, A_2, ..., A_n)$, is a set of ordered n-tuples of the form $(a_1, a_2, ..., a_n)$ such that

(1) if A_i is an atomic attribute, then $a_i \in \mathcal{D}$.

(2) if A_i is a relation scheme, then a_i is an instance of A_i.

An instance of a relation scheme is also referred to as a relation.

Let $R(S)$ be a relation scheme. $Attr(R)$ is the set of all (atomic and relation-valued) attribute names in S. $RAttr(R)$ is the set of all relation-valued attributes in S. $FAttr(R)$ is the set of all flat or atomic attributes in S. $deg(R)$ is the number of attributes in S. Henceforth, when we refer to a relation scheme, we will refer to it by its name alone. *e.g.*, R instead of $R(S)$.

Let r be an instance of R and let $t \in r$ (a tuple in relation r). If $A \in Attr(R)$ then $t[A]$ is the value of t in the column corresponding to A. If $B \subseteq Attr(R)$ then $t[B] = t[A_1]t[A_2]...t[A_m]$ where $A_i \in B \ \ (1 \leq i \leq m)$.

c is a condition on R if
(a) $c = \phi$
(b) $c = a\theta b$ where,(i) a is an atomic attribute of R or an atomic value and b is an atomic attribute, a and b have compatible domains and $\theta \in \{<, >, \leq, \geq, =, \neq\}$. (ii) a and b are relation-valued attributes of R and $\theta \in \{\subset, \subseteq, =, \neq, \supseteq, \supset\}$. (iii) b is a relation-valued attribute of R and a is a tuple in some instance of b and $\theta \in \{\in, \notin\}$.
(c) $c1$ and $c2$ are two conditions on R and $c = c1 \wedge c2$ or $c = c1 \vee c2$ or $c = \neg c1$.

If t is a tuple in some $r \in R$, then
(a) If $c = \phi$ then $c(t) = true$
(b) If $c = a\theta b$ then $c(t)$ =(1) $t[a]\theta t[b]$ if a and b are both attributes (2) $a\theta t[b]$ if only b is an attribute and (3) $t[a]\theta b$ if only a is an attribute.
(c) $c(t) = c1(t) \wedge c2(t)$, $c1(t) \vee c2(t)$ and $\neg c1(t)$ when $c = c1 \wedge c2$, $c = c1 \vee c2$ and $c = \neg c1$ respectively.

2.3 The Nested Relational Algebra

Query languages for the nested relational model were developed by Roth, Korth and Silberschatz [17], Thomas and Fischer [20], Schek and Scholl [18], and several others[14,16,10,12]. A brief set of definitions for the algebra developed by Thomas and Fischer is given below. This will be used as a basis for showing the equivalence of the recursive algebra introduced in this paper and the non-recursive algebra (the algebra of Thomas and Fischer).

Unary Operators

Unary operators are defined in the algebra of Thomas and Fischer as shown below. Figure 3 is an example of a nested relation that is used to illustrate the results of performing these operations. The respective result of each operation is shown in Figures 4 through 7.

(1) *Selection* σ

The select operator retrieves all the tuples in the relation which satisfy a certain condition.

$\sigma_c(r) = \{t \mid t \in r \mid c(t) = \ true\}$
where c is a condition on the attributes of R

(2) *Projection* π

This operator 'projects' out the columns corresponding to the attributes in the set A.

$\pi_A(r) = \{t \mid \exists u \in r \mid t = u[A]\}$
where $A \subseteq Attr(R)$

x_1

A	B		F	
	C	D / E	G	H
a_1	c_1	e_1, e_2	g_1	h_1
			g_1	h_2
	c_2	e_1		
a_2	c_2	e_1, e_3	g_1	h_1
			g_1	h_2
a_3	c_3	e_1	g_2	h_3

Fig. 3

$\sigma_{(A=a_1)\vee(A=a_3)}(x_1)$

A	B		F	
	C	D / E	G	H
a_1	c_1	e_1, e_2	g_1	h_1
			g_1	h_2
	c_2	e_1		
a_3	c_3	e_1	g_2	h_3

Fig. 4

$\pi_{\{F\}}(x_1)$

F	
G	H
g_1	h_1
g_1	h_2
g_2	h_3

Fig. 5

(3) *Nest* ν

The nest operator, also sometimes called 'pack', groups together tuples which agree on all the attributes that are not in a given set of attributes, say B. It forms a single tuple which has a new attribute name, say A, in place of B, whose value is the set of all the B values of the tuples being grouped together.

$$\nu_{B \to A}(r) = \{t \mid \exists u \in r \mid (t[Attr(R) - B] = u[Attr(R) - B]) \wedge (t[A] = \{s[B] \mid (s \in r) \wedge (s[Attr(R) - B] = t[Attr(R) - B])\})\}$$

where $B \subseteq Attr(R)$ and A is a new attribute name

$\nu_{\{A,B \to A'\}}(x_1)$

A	A' B C	A' B D E	F G	F H
a_1	c_1	e_1, e_2	g_1	h_1, h_2
	c_2	e_1		
a_2	c_2	e_1, e_3		
a_3	c_3	e_1	g_2	h_3

Fig. 6

(4) *Unnest* μ

The unnest or 'unpack' operator does the inverse of the nest operator by 'ungrouping' or flattening out the B value of the tuples.

$$\mu_B(r) = \{t \mid \exists u \in r \mid (u[Attr(R) - B] = t[Attr(R) - B]) \wedge (t[Attr(B)] \in u[B])\}$$

where $B \subseteq Attr(R)$

$\mu_{\{B\}}(x_1)$

A	C	D E	F G	F H
a_1	c_1	e_1, e_2	g_1	h_1, h_2
a_1	c_2	e_1	g_1	h_1, h_2
a_2	c_2	e_1, e_3	g_1	h_1, h_2
a_3	c_3	e_1	g_2	h_3

Fig. 7

Binary Operators

Let r_1, $r_2 \in R$ for operators 5 through 7. These operators take two relations which have the same relation scheme and return the union, difference and intersection of the relations. The relations in Figures 8 and 9 are used to illustrate these operators and the corresponding results are in Figures 10, 11, and 12

x_2

A	B	
	C	D
a_1	c_1	d_1
	c_2	d_2
a_2	c_2	d_1
a_3	c_3	d_1

Fig. 8

x_3

A	B	
	C	D
a_1	c_3	d_2
a_3	c_3	d_1

Fig. 9

$x_2 \cup x_3$

A	B	
	C	D
a_1	c_1	d_1
	c_2	d_2
a_1	c_3	d_2
a_2	c_2	d_1
a_3	c_3	d_1

Fig. 10

(5) *Union* $\cup$

$r_1 \cup r_2 = \{t \mid t \in r_1 \vee t \in r_2\}$

(6) *Difference* $-$

$r_1 - r_2 = \{t \mid t \in r_1 \wedge t \notin r_2\}$

(7) *Intersection* $\cap$

$r_1 \cap r_2 = \{t \mid t \in r_1 \wedge t \in r_2\}$

$x_2 - x_3$

A	B	
	C	D
a_1	c_1	d_1
	c_2	d_2
a_2	c_2	d_1

Fig. 11

$x_2 \cap x_3$

A	B	
	C	D
a_3	c_3	d_1

Fig. 12

(8) *Cartesian – Product* $\times$

The result of this operation is a relation whose relation scheme has the attributes of both R_1 and R_2 (renaming is done to resolve ambiguity when R_1 and R_2 have common attribute names). The tuples are formed by concatenating tuples in r_2 to those in r_1. Figure 14 shows the result of applying this operator to the relations in Figures 8 and 13.

Let $r_1 \in R_1$ and $r_2 \in R_2$

$$r_1 \times r_2 = \{t \mid \exists u \in r_1, v \in r_2 \mid$$
$$(t[Attr(R_1)] = u[Attr(R_1)]) \wedge$$
$$(t[Attr(R_2)] = v[Attr(R_2)])\}$$

x_4

A	B		E
	C	D	
a_1	c_1 c_3	d_1 d_3	e_1
a_2	c_2	d_1	e_1

Fig. 13

$x_2 \times x_4$

A	B		A'	B'		E'
	C	D		C	D	
a_1	c_3	d_2	a_1	c_1 c_3	d_1 d_3	e_1
a_3	c_3	d_1	a_1	c_1 c_3	d_1 d_3	e_1
a_1	c_3	d_2	a_2	c_2	d_1	e_1
a_3	c_3	d_1	a_2	c_2	d_1	e_1

Fig. 14

(9) $Natural - Join$ $\bowtie$

This is essentially a cartesian product followed by a selection on the tuples that agree on the values of their common attributes. Since it can be expressed in terms of cartesian product, select, and project, the definition is omitted here.

(10) $Intersection - Join$ $\Re$

Intersection join is a slight variation of natural join. The selection is done on the tuples that agree on common flat attributes and whose relation-valued attributes have at least one element in common. Again since it is expressible in terms of the other operators defined so far its definition is omitted. In fact $\pi, \sigma, \times, \cup, -, \nu$, and μ are the only essential operators and all the other operators can be expressed in terms of these.

All of the operators defined above act only on the outermost level of a relation. Queries that involve relations which are embedded within the relation require the relation to be flattened to be able to access the subrelations.

Example 1

Consider the example database shown in Figure 2. Let us suppose that we want the set of all the shares purchased on 02/10/83 and we want them listed by their owner's name and the company of the share. We would formulate this query in the algebra of Thomas and Fischer as follows:

$$\nu_{\text{COMPANY,SHARES}\rightarrow\text{INVESTMENTS}}(\nu_{\text{PURCHASE-PRICE,DATE,NO.}\rightarrow\text{SHARES}}$$
$$(\pi_{\text{NAME,COMPANY,PURCHASE-PRICE,DATE,NO.}}$$
$$(\sigma_{\text{DATE='02/10/83'}}(\mu_{\text{SHARES}}(\mu_{\text{INVESTMENTS}}(\text{CLIENTS}))))))$$

The table shown in Figure 15 is the result of performing this query. The unnest operators, μ, transform the relation CLIENTS into a flat relation. Then, after the selection and projection have been performed on the flat relation, the nest operators, ν, restructure the flat relation back into a nested one. Thus, queries in this algebra cause data to be restructured while performing operations like select and project. This restructuring is necessary because the algebraic operators operate only at the outermost level of a relation. Hence, eventhough the first normal form has been relaxed to include relation valued attributes, operators like select, project etc. essentially treat all the components of a tuple as non-decomposable or atomic units.

NAME	INVESTMENTS			
	COMPANY	SHARES		
		PURCHASE PRICE	DATE	NO.
John Smith	XEROX	64.50	02/10/83	100
Jill Brody	EXXON	59.50	02/10/83	200
	FORD	35.50	02/10/83	200

Fig. 15

Although the query in the non-recursive algebra shown here works for this example, in general relations have to be "tagged" first before unnesting and nesting. The above query will not give us the correct result if the relation were not in hierarchical normal form (a relation is in hierarchical normal form (HNF) if (1) all or a subset of the atomic attributes form a key for the relation and (2) each of its relation-valued attributes is in HNF). The relation shown in Figure 16 is not in HNF. Now, let us suppose that we want to select all the tuples that have a 0 or a 1 in attribute D. The result that we expect is shown in Figure 17 and the result that we actually get by unnesting on B, selecting tuples that have a 0 or a 1 in D, and then renesting on C and D is shown in Figure 18.

r_1

A	B	
	C	D
		E
a_1	c_1	0
a_1	c_2	1
	c_3	2

Fig. 16

$\sigma(r_1(B_{(0\in D)\vee(1\in D)}))$

A	B	
	C	D
		E
a_1	c_1	0
a_1	c_2	1

Fig. 17

$\nu_{C,D\to B}(\sigma_{(0\in D)\vee(1\in D)}(\mu_B(r_1)))$

A	B	
	C	D
		E
a_1	c_1	0
	c_2	1

Fig. 18

To get the correct result in this case, we would have to first tag or index the tuples of the relation before unnesting. The Index operator, defined originally by Van Gucht and Fischer [24], is basically a way of tagging each row of a tuple with a unique value. It adds a new column with attribute name I, to the relation and the values in this column are just the tuples in the corresponding rows of the original relation.

Definition

$$Index(r) = \{t \mid \exists t_r \in r \,|(t[Attr(R)] = t_r[Attr(R)]) \\ (t[I] = \{t_r\})\}$$

This operator can be expressed in terms of nest, select and join of Thomas and Fischer. $Index(r) = \nu_{A'\to I}\sigma_{A'=A}(r \times r)$, where A is set of attributes of R and A' is the set of new attributes in $r \times r$. Tagging the tuples of a relation is required to save information about

how tuples were grouped together before an unnest was performed. Each of the recursive operators can be expressed in terms of a combination of the non-recursive ones including the non-recursive nest and unnest. It has been shown by Van Gucht [22] that unnesting a relation on a relation-valued attribute and then nesting it back doesn't always give us back the original relation; which is why it is necessary to tag the tuples before unnesting.

The correct expression for the query in Example 1 is then

$$\pi_{\text{NAME,INVESTMENTS}}(\nu_{\text{COMPANY,SHARES}\rightarrow\text{INVESTMENTS}}$$
$$(\pi_{\text{NAME,COMPANY,SHARES,I}_2}(\nu_{\text{PURCHASE-PRICE,DATE,NO.}\rightarrow\text{SHARES}}$$
$$(\pi_{\text{NAME,COMPANY,PURCHASE-PRICE,DATE,NO.,I}_1\text{,I}_2}$$
$$(\sigma_{\text{DATE='02/10/83'}}(\mu_{\text{SHARES}}(Index(\mu_{\text{INVESTMENTS}}(Index(\text{CLIENTS }))))))))))$$

where, I_1 and I_2 are the index columns added by the two $Index$ operations.

Queries in the non-recursive algebra for nested relations, like the one in the previous example, are slow and complicated since accessing the inner levels of a relation requires unnesting and nesting. The nested relational model is an extension of the traditional relational model by allowing values in a relation to be relation-valued; but the non-recursive algebra for nested relations is not an extension of the algebra for flat relations to the same extent. The operators in the non-recursive algebra have not been extended to operate on the subrelations of a relation. The only way in which an operation can be performed on a subrelation is by flattening the relation using the unnest operation repeatedly until the attributes of the subrelation are at the outermost level.

3. Recursive Algebra

3.1 The Recursive Algebra

The algebra presented in this section has operators that are recursively defined which can hence apply themselves repeatedly to the subrelations at the different levels of a relation. This eliminates the need for a special operator to serve as a navigator since each operator can itself traverse the different levels of a nested relation. It also reduces the number of nest

and unnest operations in a query. Nest and unnest are, however, still required since queries may require the structure of relations to be changed; although they are not necessary for accessing the inner levels of a relation. In the examples given in this section, the relations x_1, x_2, x_3, x_4 refer to the relations shown in Figures 3, 8, 9 and 13 respectively.

(1) *Selection* σ

Performing selection on a nested relation when the selection condition involves attributes that are not at the outermost level, as in Example 1, involves a series of unnests with indexing and nests. The selection operator introduced here can operate on attributes at any level without having to flatten the relation first.

Let R be a relation scheme. Then, L is a 'select list' of R if
(i) L is empty
(ii) L is of the form $(R_{1c_1}L_1, R_{2c_2}L_2, ..., R_{nc_n}L_n) \quad (1 \leq n \leq \mid RAttr(R) \mid)$
where each R_i is a relation-valued attribute of R, c_i is a condition on R_i, and L_i is a select list of R_i.

Let r be a relation with relation scheme R.

$$(i) \quad \sigma(r_c) = \{t \in r \mid c(t) = true\}$$

$$(ii) \quad \sigma(r_c(R_{1c_1}L_1, R_{2c_2}L_2, ..., R_{nc_n}L_n))$$
$$= \{t \mid \exists t_r \in r \mid$$
$$t[Attr(R) - (R_1, R_2, ..., R_n)] = t_r[Attr(R) - (R_1, R_2, ..., R_n)]$$
$$\wedge (c(t_r) = true)$$
$$\wedge (t[R_1] = \sigma\left((t_r[R_1])_{c_1} L_1\right) \neq \phi)$$
$$\vdots$$
$$\wedge (t[R_n] = \sigma\left((t_r[R_n])_{c_n} L_n\right) \neq \phi)\}$$

where $L = (R_{1c_1}L_1, R_{2c_2}L_2, ..., R_{nc_n}L_n)$ is a select list of R and the L_i's are (possibly empty) select lists of R_i's and c is a condition on R.

(2) *Projection* π

Projection in the non-recursive algebra, like selection, operates only at the outermost level in a nested relation. In the stock-broker database example, if we wanted the purchase-dates of all the shares, listed by the company and owner's name, we would have to perform two unnests before being able to access the attribute PURCHASE-DATE and then two nests to get the result in the desired form. The recursive definition of projection given here allows us to perform projection on attributes at all levels without restructuring.

Let R be a relation scheme. Then, L is a 'project list' of R if

(i) L is empty

(ii) L is of the form $(R_1L_1, ..., R_nL_n)$

where R_i is an attribute of R and L_i is a project list of R_i (L_i is empty if R_i is an atomic-attribute).

Let r be a relation with relation scheme R.

(i) $\pi(r) = r$

(ii) $\pi((R_1L_1, ..., R_nL_n)r) = \{t \mid \exists t_r \in r \mid$

$$t = \pi(L_1(t_r[R_1]))...\pi(L_n(t_r[R_n]))\}$$

where $(R_1L_1, ..., R_nL_n)$ is a project list of R.

$\sigma(x_1(B_{c=c_2}))$

A	B		F	
	C	D / E	G	H
a_1	c_2	e_1	g_1	h_1
			g_1	h_2
a_2	c_2	e_1	g_1	h_1
		e_3	g_1	h_2

Fig. 19

$\pi((A, B(D))x_1)$

A	B
	D
	E
a_1	e_1
	e_2
	e_1
a_2	e_1
	e_3
a_3	e_1

Fig. 20

With the selection and projection defined as above, the query for Example 1 can be expressed as follows.

$\pi((\text{NAME,INVESTMENTS})(\sigma(\text{CLIENTS}(\text{INVESTMENTS}(\text{SHARES}_{\text{DATE='02/10/83'}})))))$

(3) *Nest* ν

The operators nest and unnest restructure or change the way tuples in a relation are grouped together in a nested relation. In the stock-broker example, if we wanted to restructure the sub-relation INVESTMENTS, so that shares are grouped by the PURCHASE-DATE instead of by COMPANY, we would have to perform two unnests and two nests as shown below.

Example 2

$\nu_{\text{PURCHASE-DATE,SHARES}\rightarrow\text{INVESTMENTS}}(\nu_{\text{COMPANY,PURCHASE-PRICE,NO.}\rightarrow\text{SHARES}}$
$(\mu_{\text{SHARES}}(\mu_{\text{INVESTMENTS}}(\text{CLIENTS}))))$

Thus, eventhough we want to restructure only the subrelation INVESTMENTS, we have to unnest the entire relation CLIENTS on INVESTMENTS and then nest it again after restructuring. It would be convenient to be able to restructure only the subrelation that we are interested in without affecting any of the other attributes in the relation. The nest and unnest operators defined below enable us to do precisely this.

Let R be a relation scheme. L is a 'nest list' of R if
(i) L is of the form $(R_1, ..., R_n) \rightarrow A$ where each $R_i \in Attr(R)$ and A is a new attribute name s.t. $A \notin Attr(R)$.
(ii) L is of the form $R_i(L_i)$ where $R_i \in RAttr(R)$ and L_i is a nest list of R_i.

Let r be a relation with relation scheme R.

$$(i)\ \nu(r(R_1,...,R_n) \rightarrow A) = \{t \mid \exists t_r \in r \mid$$
$$(t[Attr(R) - (R_1,...,R_n)] = t_r[Attr(R) - (R_1,...,R_n)]) \wedge$$
$$t[A] = \{s \mid \exists p \in r \mid$$
$$(p[Attr(R) - (R_1,...,R_n)] = t[Attr(R) - (R_1,...,R_n)])$$
$$\wedge (s = p[R_1,...,R_n])\}\}$$

$$(ii)\ \nu(r(R_i(L_i)) \rightarrow A) = \{t \mid \exists t_r \mid$$
$$(t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$$
$$\wedge (t[R_i] = \nu(t_r[R_i](L_i) \rightarrow A)\}$$

(4) *Unnest* μ

Let R be a relation scheme. L is an 'unnest list' of R if

(i) L is of the form R_i where $R_i \in RAttr(R)$

(ii) L is of the form $R_i(L_i)$ where $R_i \in RAttr(R)$ and L_i is an unnest list of R_i.

Let r be a relation with relation scheme R.

$$(i)\ \mu(r(R_i)) = \{t \mid \exists t_r \in r \mid$$
$$(t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$$
$$\wedge (t[Attr(R_i)] \in t_r[R_i])\}$$
$$(ii)\ \mu(r(R_i(L_i))) = \{t \mid \exists t_r \in r \mid$$
$$(t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$$
$$\wedge t[R_i] = \mu(t_r[R_i](L_i))\}$$

The query in Example 2 can be expressed in terms of the nest and unnest that we have just defined, as follows.

$\nu((\mu(\text{CLIENTS(INVESTMENTS(SHARES))}))(\text{INVESTMENTS(COMPANY,PURCHASE-PRICE,NO.)}) \rightarrow$ SHARES)

Note: Indexing would be necessary in this example too if the relation were not in HNF.

$\nu(x_1(F(H)) \rightarrow H')$

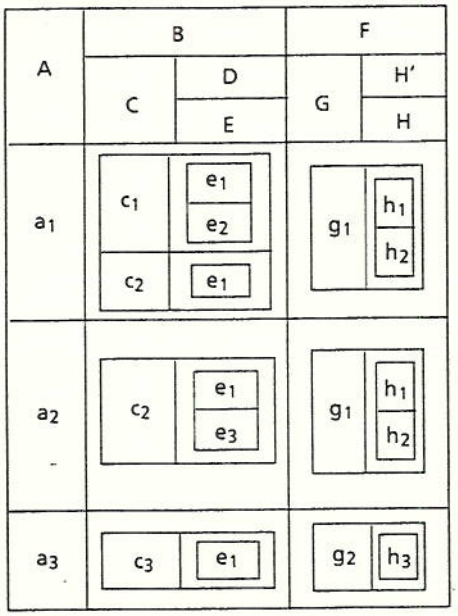

Fig. 21

$\mu(x_1(B(D)))$

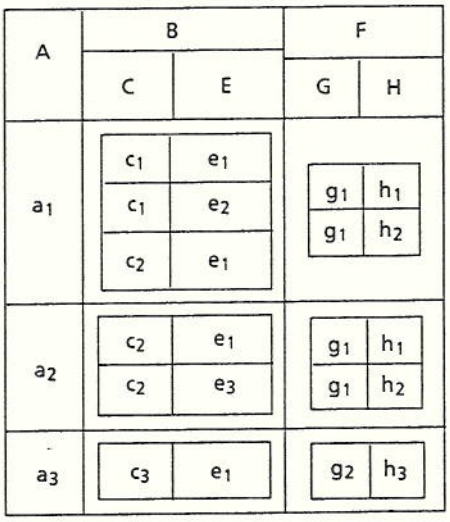

Fig. 22

(5) *Union* $\cup$

Union, difference, and intersection usually involve entire tuples. However, in some cases, a user may wish to perform one of these operations between two relations such that the tuples of certain subrelations are also taken into consideration. For example, if the tuple (John,(x,y)) is added to a relation that contains the tuple (John,(z)), a union that involves just the entire tuples in the relation would give us a relation that has the tuples (John,(x,y)) and (John,(z)) in it. It might be more desirable in this case to have the union operation return (John, (x,y,z)) as the result. Deshpande and Larson[6] define union, difference and intersection in a way that preserves the hierarchical normal form of relations. A relation is in hierarchical normal form if the key contains only atomic valued attributes. This assumption is not made in this paper. However, since users may wish to use either or both types of unions, intersections and joins, two different operators are defined for each of these operations.The operators which consider whole tuples are the same as those of Thomas and Fischer and the ones

which preserve the functional dependency between the key attributes and the other attributes of a relation are the same as those introduced by Deshpande and Larson with a small variation. Deshpande and Larson assume that the atomic attributes of a relation form a key. Hence, for instance, when a union is performed, two tuples that agree on all the atomic attributes are merged together at their relation- valued attributes. Since this assumption is not made here, while performing a union, tuples that agree on all the atomic attributes and all the key attributes are merged at the relational-valued attributes that are not in the key. The result of performing this type of union on the relations in Figures 8 and 9 is shown below.

$\cup^e(x_2, x_3)$

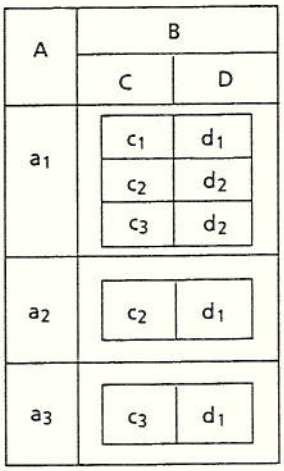

Fig. 23

The operators $\cup$, $-$, and $\cap$ are defined just as they arein the non-recursive algebra.

For items 5 through 7 let r_1 and r_2 be two relations with relation scheme R. Let $k(R) = key(R) \cup FAttr(R)$ and let $m(R) = RAttr(R) - k(R)$.

$\cup^e$, $-^e$ and $\cap^e$ are defined recursively.

(1) $\cup(r_1, r_2) = \{t \mid (t \in r_1) \vee (t \in r_2)\}$

(2) $\cup^e(r_1, r_2) = \{t \mid ((t \in r_1) \wedge (\forall t_2 \in r_2, t_2[k(R)] \neq t[k(R)]))$

$\vee ((t \in r_2) \wedge (\forall t_1 \in r_1, t_1[k(R)]) \neq t[k(R)]))$

$\vee (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$

$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$

$\wedge (t[R_1] = \cup^e(t_1[R_1], t_2[R_1]))$

$\vdots$

$\wedge (t[R_l] = \cup^e(t_1[R_l], t_2[R_l])))$

$where\ R_i \in m(R)\ (1 \leq i \leq l)\}$

(6) *Difference* $-$

(1) $-(r_1, r_2) = \{t \mid (t \in r_1) \wedge (t \notin r_2)\}$

(2) $-^e(r_1, r_2) = \{t \mid ((t \in r_1) \wedge (\forall t_2 \in r_2, t_2[k(R)] \neq t[k(R)]))$
$\vee (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$
$(t[k(R)] = t_1[k(R)] = t_2[k(R)]) \wedge (t_1 \neq t_2)$
$\wedge (t[R_1] = -^e(t_1[R_1], t_2[R_1]))$
$\vdots$
$\wedge (t[R_l] = -^e(t_1[R_l], t_2[R_l])))$ *where* $R_i \in m(R)\ (1 \leq i \leq k)\}$

(7) *Intersection* $\cap$

(1) $\cap(r_1, r_2) = \{t \mid (t \in r_1) \wedge (t \in r_2)\}$

(2) $\cap^e(r_1, r_2) = \{t \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$
$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$
$\wedge (t[R_1] = \cap^e(t_1[R_1], t_2[R_1]))$
$\vdots$
$\wedge (t[R_l] = \cap^e(t_1[R_l], t_2[R_l])))$ *where* $R_i \in m(R)\ (1 \leq i \leq l)\}$

(8) *Cartesian-Product* $\times$

In the algebra of Thomas and Fischer, joins and cartesian-products, like all the other algebraic operations, are done at the outermost level of the relations being joined. However, in some cases, it may be convenient to be able to join the tuples of one relation to tuples of a relation valued attribute nested deep inside the structure of the relation scheme. For example, consider the relations shown in Figure 2. Let us suppose that we want the total net value of all the shares that John has. Obviously, a join must be performed with the relation STOCK-DATA since it has the current price of each share and we would like to perform this join on the sub-relation INVESTMENTS without having to unnest and nest. This is not possible in the algebra of Thomas of

Fischer where joins are performed only between two relations and not between a sub-relation and a relation. The definitions for these operations given below allow joins and cross-products to be performed between a relation or a relation-valued attribute and another relation. Since it does not make any sense to perform these operations between two sub-relations, these operations are not symmetric, *i.e.*, one operand can be a relation or a sub-relation while the other has to be a relation.
Let R be a relation scheme.

L is an 'access path' of R if
(i) L is empty.
(ii) L is of the form $R_i(L_i)$ where R_i is a relation-valued attribute of R and L_i is an access path of R_i.

Let r and q be two relations with relation schemes R and Q respectively.

$$(i) \times (r, q) = \{t \mid \exists t_r \in r, t_q \in q \mid (t[Attr(R)] = t_r) \wedge (t[Attr(Q)] = t_q)\}$$

$$(ii) \times (r(R_i(L_i)), q) = \{t \mid \exists t_r \in r \mid (t[R_i] = \times(t_r[R_i](L_i), q) \wedge (t[Attr(R) - R_i] = t_r[Attr(R) - R_i])\}$$

where $(R_i(L_i))$ is an access path of R.

We assume that common attributes in R and Q are renamed in order to resolve ambiguity.

(9) *Equi-Join* $\bowtie_=$

Let R and Q be two relation schemes. L is an equi-join list of R and Q if
(i) L is of the form $(R_1, ..., R_n)$ where each $R_i \in Attr(R) \cap Attr(Q)$.
(ii) L is of the form $R_i(L_i)$ where R_i is a relation-valued attribute of R and L_i is an equi-join list of R_i and Q.
Let r and q be two relations with relation schemes R and Q respectively.

$$(i) \bowtie_{=} (r(R_1, ..., R_n), q) \quad =$$

$$\{t \mid \exists t_r \in r, t_q \in q \mid$$

$$(t_r[R_1, ..., R_n] = t_q[R_1, ..., R_n] = t[R_1, ...R_n])$$

$$\wedge (t[Attr(R) - (R_1, ..., R_n)] = t_r[Attr(R) - (R_1, ..., R_n)])$$

$$\wedge (t[Attr(Q) - (R_1, ..., R_n)] = t_q[Attr(Q) - (R_1, ..., R_n)])\}$$

$$(ii) \bowtie_{=} (r(R_i(L_i)), q) =$$

$$\{t \mid \exists t_r \in r \mid$$

$$(t[R_i] = \bowtie_{=} (t_r[R_i](L_i), Q) \neq \phi)$$

$$\wedge (t[Attr(R) - R_i] = t_r[Attr(R) - R_i])\}$$

x_5

W	S		V
	T	A	
w_1	t_1	a_1	v_1
w_2	t_1	a_2	v_1

Fig. 24

$\bowtie_{=}(x_5(S(A)), x_2)$

W	S				V
	T	A	B		
			C	D	
w_1	t_1	a_1	c_1	d_1	v_1
			c_2	d_2	
w_2	t_1	a_2	c_2	d_1	v_1

Fig. 25

Expressions in the recursive algebra are defined as follows. Any expression denoting a constant relation is an expression. Any combination of the operators defined above applied to an expression is an expression. Relations can have empty sets as values for set-valued attributes. In particular, we will denote the relation with a single relation-valued attribute (the only attribute) in its relation scheme and with only one tuple which contains the empty set by E(A), where A is the attribute in the relation-scheme.

The following example shows the advantage of having a join operation that allows

joins to be performed on a subrelation of a relation without any unnesting or nesting.

Example 3
Let us suppose that we want the investments that clients have in stock that is traded in London, listed by the client's name and address. This query can be expressed in the non-recursive algebra as follows.

$\nu_{\text{COMPANY,SHARES}\rightarrow\text{INVESTMENTS}}(\mu_{\text{INVESTMENTS}}(\text{CLIENTS}) \bowtie$
$\pi_{\text{COMPANY}}(\sigma_{\text{LONDON}\in\text{EXCHANGES-TRADED}}(\text{STOCK-DATA})))$

The same query can be written in the recursive algebra without any restructuring operators as follows.

$\bowtie_{=}(\text{CLIENTS (INVESTMENTS (COMPANY))} ,\pi((\text{COMPANY})$
$(\sigma(\text{STOCK-DATA}_{\text{LONDON}\in\text{EXCHANGES-TRADED}}))))$

3.2 Related Work

Deshpande and Larson [6] have an algebra for nested relations in which queries can be formulated such that relations don't have to be flattened in order to access and manipulate data at interior levels of relations. This is done by means of a subrelation constructor. This constructor allows relations to be accessed and modified at all levels by creating new subrelations while traversing the different levels of a relation. Although this construct provides the algebra with more navigational ability, it is not very convenient for the user to have to introduce new relations and names for these new relations within the query. Also, since all the other operators operate only at the outermost level, the subrelation constructor is needed everytime an operation has to be performed at an interior level.

Schek and Scholl [18] introduced a new algebra for nested relations that enables the user to traverse the database without nesting and unnesting. However, their algebra has several limitations. Firstly, only one operator, *viz.* π, is used as a "navigator". For instance if the user wants to unnest a subrelation nested deep inside a relation, he would have to write the query in terms of a series of projections and an unnest. Secondly, renaming

is necessary in a number of situations, which is an inconvenience for the user. Thirdly, union, difference and selection do not involve the subtuples of tuples. Finally, the algebra is rather complicated.

Roth, Korth and Batory [16] have proposed an extension to SQL for nested relations which allows nested application of queries. However, no formal semantics for their language is given.

4. Equivalence of the two algebras

In this section, we show that the recursive and the non-recursive algebras are each expressible in terms of the other and hence equivalent to each other in terms of their expressive powers. We first show that the non-recursive selection can be expressed in terms of the recursive selection and that the recursive selection can be expressed in terms of a subset of the basic non-recursive operators. The equivalence proofs for projection, nest, unnest, and cartesian product are similar and are omitted for brevity. We then give outlines of proofs for union and difference.

Let us denote the selection operator that has just been introduced by σ', to distinguish it from the σ of Thomas and Fischer . Similarly, the new operators for projection, nest, unnest, etc. will be denoted by π', ν', μ' and so on, while π, ν, μ etc. will denote the corresponding operators of Thomas and Fischer. To establish the equivalence of the two algebras we consider only the basic operators (in both the algebras), *i.e.,* we do not have to show, for example, that $\bowtie$ is expressible in the recursive algebra if we show that σ, π, and $\times$ are (since $\bowtie$ can be expressed in terms of these operators).

Lemma 1 If r is a relation with relation scheme R, $(R_{1\,c_1}L_1)$ a select list on R, and c a condition on R, then
$\sigma'(r_c(R_{1\,c_1}L_1)) = \pi_{Attr(R)}(\nu_{Attr(R_1)\rightarrow R_1}(\sigma'(\mu_{R_1}(Index(\sigma'(r_c)))_{c_1}L_1)))$

Proof:

$$\begin{aligned}
\sigma'(r_c(R_{1\,c_1}L_1)) &= \{t \mid \exists t_r \in r \mid (t[Attr(R) - R_1] = t_r[Attr(R) - R_1]) \wedge \\
&\qquad (c(t_r) = true) \wedge (t[R_1] = \sigma'((t_r[R_1])_{c_1} L_1))\} \\
&= \{t \mid \exists t_{r'} \in r' \mid (t[Attr(R) - R_1] = t_{r'}[Attr(R) - R_1]) \wedge \\
&\qquad (t[R_1] = \sigma'((t_{r'}[R_1])_{c_1} L_1)) \text{ where } r' = \{t_r \in r \mid c(t_r) = true\}\} \\
&= \{t \mid \exists t_{r'} \in r' \mid (t[Attr(R) - R_1] = t_{r'}[Attr(R) - R_1]) \wedge \\
&\qquad (t[R_1] = \sigma'((t_{r'}[R_1])_{c_1} L_1)) \text{ where } r' = \sigma'(r_c)\} \\
&= \sigma'(\sigma'(r_c)(R_{1\,c_1}L_1)) \\
&= \pi_{Attr(R)}(\sigma'(Index(\sigma'(r_c))_{c_1} L_1)) \\
&= \pi_{Attr(R)}(\nu_{Attr(R_1)\to R_1}(\sigma'(\mu_{R_1}(Index(\sigma'(r_c)))_{c_1} L_1)))
\end{aligned}$$

Lemma 2 If r is a relation with relation scheme R, $L = (R_{1\,c_1}L_1, \ldots R_{n\,c_n}L_n)$ a select list on R, and c a condition on R, then
$\sigma'(r_c(R_{1\,c_1}L_1, \ldots, R_{n\,c_n}L_n)) = P_n(r_c, L)$ where P_n is defined as follows.

$$P_0(r_c, L) = \sigma_c(r)$$
$$P_i(r_c, L) = \pi_{Attr(R)}(\nu_{Attr(R_i)\to R_i}(\sigma'(\mu_{R_i}(Index(P_{i-1}(r_c, L)))_{ci} L_i)))$$

Proof: By induction on n.

Basis: $n = 0$
$\sigma'(r_c) = \{t \in r \mid c(t) = true\} = \sigma_c(r) = P_0(r_c, L)$

Induction Hypothesis: Let Lemma 2 be true for all $k \leq n - 1$
Let $r' = \sigma'(r_c(R_{1\,c_1}L_1, \ldots, R_{n-1\,c_{n-1}}L_{n-1}))$
Hence, by induction hypothesis, $r' = P_{n-1}(r_c, (R_{1\,c_1}L_1, \ldots, R_{n-1\,c_{n-1}}L_{n-1})) = P_{n-1}(r_c, L)$

$$\begin{aligned}
\text{Now, } \sigma'(r_c(R_{1\,c_1}L_1, \ldots, R_{n\,c_n}L_n)) &= \sigma'(r'(R_{n\,c_n}L_n)) \\
&= \pi_{Attr(R)}(\nu_{Attr(R_n)\to R_n}(\sigma'(\mu_{R_n}(Index(r'))_{c_n} L_n))) \text{ by Lemma 1} \\
&= \pi_{Attr(R)}(\nu_{Attr(R_n)\to R_n}(\sigma'(\mu_{R_n}(Index(P_{n-1}(r_c, L)))_{c_n} L_n))) \\
&= P_n(r_c, L)
\end{aligned}$$

Definition If L is a select list on a relation scheme R, then the *depth* of L, denoted $d(L)$ is defined as follows:
(i) If L is empty, $d(L) = 0$,
(ii) If $L = (R_{1\,c_1}L_1, ..., R_{n\,c_n}L_n)$, $d(L) = 1 + max\{d(L_1), ..., d(L_n)\}$.

Theorem 1 σ can be expressed in terms of σ' and σ' can be expressed in terms of σ, π, ν, μ and $\times$.

Proof: Let r be a relation with relation scheme R

$\sigma_c(r) = \{t \in r \mid c(t) = true\} = \sigma'(r_c)$

Hence σ can be expressed in terms of σ'

Let $L = (R_{1\,c_1}L_1, ..., R_{n\,c_n}L_n)$ be a select list on R.

Case 1 $d(L) = 0$ *implies* $L = \phi$

$\sigma'(r_c) = \{t \in r \mid c(t) = true\} = \sigma_c(r)$ by definition.

Case 2 $d(L) = 1$

Case 2a $n = 1$ *implies* $L = (R_{1\,c_1})$

$$\sigma'(r_c(R_{1\,c_1})) = \pi_{Attr(R)}(\nu_{Attr(R_1)\to R_1}(\sigma'(\mu_{R_1}(Index(\sigma'(r_c)))_{c_1}))) \quad \text{by Lemma 1}$$
$$= \pi_{Attr(R)}(\nu_{Attr(R_1)\to R_1}(\sigma_{c_1}(\mu_{R_1}(Index(\sigma_c(r)))))) \quad \text{by case 1}$$

Case 2b $n > 1$ *imp* $L = (R_{1\,c_1}, ..., R_{n\,c_n})$

Induction hypothesis: Let Theorem 1 be true when $length(L) < n$

$$\sigma'(r_c(R_{1\,c_1}, ..., R_{n\,c_n})) = P_n(r_c, L) \quad \text{by Lemma 2}$$
$$= \pi_{Attr(R)}(\nu_{Attr(R_n)\to R_n}(\sigma'(\mu_{R_n}(Index(P_{n-1}(r_c, L)))_{c_n})))$$
$$= \pi_{Attr(R)}(\nu_{Attr(R_n)\to R_n}(\sigma_{c_n}(\mu_{R_n}(Index(P_{n-1}(r_c, L))))))$$

Now, $P_{n-1}(r_c, L) = \sigma'(r_c(R_{1\,c_1}, ..., R_{n-1\,c_{n-1}}))$

Since $length(R_{1\,c_1}, ..., R_{n-1\,c_{n-1}}) = n-1 < n$, P_{n-1} is expressible in terms of σ, π, ν, μ and $Index$, by the induction hypothesis. Hence, $\sigma'(r_c(R_{1\,c_1}, ..., R_{n\,c_n}))$ is expressible in terms of these operators and so from this and Case 2a, Theorem 1 follows for Case b.

Case 3 $d(L) = k > 1$
Case 3a $n = 1$ *implies* $L = (R_{1\,c_1} L_1)$
Induction Hypothesis Let Theorem 1 be true when $d(L) < k$
$\sigma'(R_{1\,c_1} L_1) = \pi_{Attr(R)}(\nu_{Attr(R_1) \rightarrow R_1}(\sigma'(\mu_{R_1}(Index(\sigma_c(r)))_{c_1} L_1)))$ by Lemma 1 and Case 1
Since $d(L_1) = k - 1 < k$, $\sigma'(\mu_{R_1}(Index(\sigma_c(r)))_{c_1} L_1)$ can be expressed in terms of the non-recursive operators (by the hypothesis). From this result and Case 2a, Theorem 1 follows for Case 3a.

Case 3b $n > 1$ imp $L = (R_{1\,c_1} L_1, ..., R_{n\,c_n} L_n)$
Induction hypothesis Let Theorem 1 be true for all L s.t. $d(L) < k$
By Lemma 2,
$\sigma'(r_c(R_{1\,c_1} L_1, ..., R_{n\,c_n} L_n)) = \pi_{Attr(R)}(\nu_{Attr(R_n) \rightarrow R_n}(\sigma'(\mu_{R_n}(Index(P_{n-1}(r_c, L)))_{c_n} L_n)))$

By induction hypothesis, $\sigma'(\mu_{R_n}(Index(P_{n-1}(r_c, L)))_{c_n} L_n)$ can be expressed in terms of the non-recursive operators and $P_{n-1}(r_c, L)$. Therefore, if $P_{n-1}(r_c, L)$ can be expressed in terms of the non-recursive operators then so can $P_n(r_c, L)$. But $P_0(r_c, L) = \sigma'(r_c) = \sigma_c(r)$. Hence by induction, $P_n(r_c, L)$ can be expressed in terms of the non-recursive operators and Theorem 1 follows for Case 3b.

Along similar lines it is possible to show that the recursive operators π', ν', μ' and $\times'$ can be expressed as a combination of the basic non-recursive ones. (Note: $Index$ can be expressed in terms of the basic non-recursive operators σ, ν and $\times$.). The proofs for these are omitted since they are similar to the previous proof. The only major difference in the expressions in the non-recursive algebra which express the corresponding operators of the recursive algebra and the non-recursive expression for selection is that special consideration must be given for tuples that have empty sets as values corresponding to relation-valued attributes. Unnesting on such relation-valued attributes causes these tuples to be lost. Figure 27 shows the result of unnesting the relation in Figure 26 on B (after indexing it),

nesting on D and then nesting it on C and E which is different from the result of performing the recursive nest, which is shown in Figure 28.

r_2

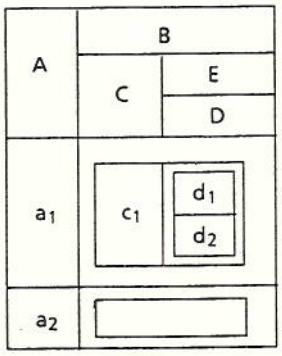

Fig. 26

$\pi_{A,B}(\nu_{C,E\rightarrow B}(\nu_{D\rightarrow E}(\mu_B(Index(r_2)))))$

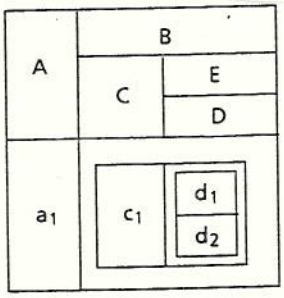

Fig. 27

$\nu(r_2(B(D)) \rightarrow E)$

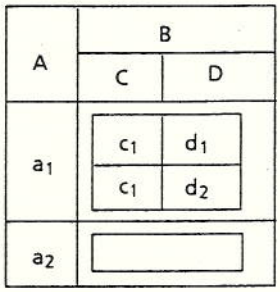

Fig. 28

In such cases, at every stage of nesting (corresponding to a previous unnest operation), tuples that have been lost as a result of unnesting should be added to the relation. For instance, for the recursive nest operator, the equivalent of Lemma 1 for selection will be as follows. If r is a relation with relation scheme R, $R_i(L_i)$ a nest list on R, and A a new attribute name, then $\nu'(r(R_i(L_i)) \rightarrow A) = r' \cup \sigma_{R_i=\phi}(r)$
where $r' = \pi_{Attr(R)}(\nu_{Attr(R_1)\rightarrow R_1}(\nu'(\mu_{R_1}(Index(r)))L_i \rightarrow A))$

The proofs for $\cup^e$ and $-^e$ are slightly different and outlines of their proofs are given below. In the case of difference, tuples can be lost not only as a result of unnesting on empty sets but also in some cases when taking the difference after unnesting. See Figures 29-32. For this we need to be able to create tuples that have empty sets as values and there is really no way of doing this in the non-recursive algebra of Thomas and Fischer. However, if we augument their algebra with constant expressions that denote relations over a single attribute that have only one tuple which has the empty set as the value for that attribute, as specified in section 3.1 for the recursive algebra, it will be possible to express the recursive operators in terms of the non-recursive ones.

Lemma 3 Let r_1 and r_2 be two relations with relation scheme $R(A_1, ..., A_n)$. Let $m(R)$, the set of non-key relation-valued attributes of R, be such that $\mid m(R) \mid = 1$. Without loss of generality we can assume that $\{A_n\} = m(R)$ and $\{A_1, ..., A_{n-1}\} = k(R)$. Then,

$$\cup^e(r_1, r_2) = (r_1 - \pi_{A_1,\ldots A_n}(r')) \cup (r_2 - \pi_{A'_1,\ldots A'_n}(r')) \cup (\pi_{k(R)\cup A_n}(\sigma_{A_n=A'_n=\phi}(r'))) \cup$$
$$\pi_{k(R)\cup A_n}(\nu_{Attr(A_n)}(\cup^e(\mu_{A_n}(\pi_{k(R)\cup I\cup A_n}(Index(r'))),$$
$$\mu_{A'_n}(\pi_{k(R)'\cup I\cup A'_n}(Index(r'))))))$$

where $r' = \sigma_{k(R)=k(R)'}(r_1 \times r_2)$

Note: When the cartesian product of two relations r_1 and r_2 with the same relation scheme is taken, we assume that all the attributes $A_i \in R$ corresponding to r_2 are renamed to A'_i respectively in the scheme of the result. Similarly, $k(R)'$ is the set of attributes A'_i such that $A_i \in k(R)$. The attribute I denotes the new attribute that is added to a relation scheme when the *Index* operator is applied to it.

Proof

$$\cup^e(r_1, r_2) = \{t \mid ((t \in r_1) \wedge (\forall t_2 \in r_2, t_2[k(R)] \neq t[k(R)]))$$
$$\vee ((t \in r_2) \wedge (\forall t_1 \in r_1, t_1[k(R)]) \neq t[k(R)]))$$
$$\vee (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid \quad \text{by definition}$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$$
$$\wedge (t[R_1] = \cup^e(t_1[R_1], t_2[R_1])))\}$$

Now, $\{t \mid t \in r_1 \mid \forall t_2 \in r_2, t[k(R)] \neq t_2[k(R)]\} = r_1 - \pi_{A_1,\ldots,A_n}(r')$

and $\{t \mid t \in r_2 \mid \forall t_1 \in r_1, t[k(R)] \neq t_1[k(R)]\} = r_2 - \pi_{A'_1,\ldots,A'_n}(r')$

Therefore,
$$\cup^e(r_1, r_2) = (r_1 - \pi_{A_1,\ldots,A_n}(r')) \cup (r_2 - \pi_{A'_1,\ldots,A'_n}(r')) \cup$$
$$\{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$$
$$\wedge (t[R_1] = \cup^e(t_1[R_1], t_2[R_1]))\}$$

The expression on the right hand side of the above equation is the union of three different terms. Let r'' denote the last term.

$$
\begin{aligned}
r'' &= \{t \mid \exists t' \in (r_1 \times r_2) \mid (t[k(R)] = t'[k(R)] = t'[k(R)'])\\
&\quad \wedge (t[A_n] = \cup^e(t'[A_n], t'[A'_n]))\}\\
&= \{t \mid \exists t' \in \sigma_{k(R)=k(R)'}(r_1 \times r_2) \mid (t[k(R)] = t'[k(R)])\\
&\quad \wedge (t[A_n] = \cup^e(t'[A_n], t'[A'_n]))\}\\
&= \{t \mid \exists t_1 \in \pi_{k(R)\cup I\cup A_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2))),\\
&\quad \exists t_2 \in \pi_{k(R)'\cup I\cup A'_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2))) \mid\\
&\quad (t[k(R)] = t_1[k(R)] = t_2[k(R)'])\wedge\\
&\quad (t[A_n] = \cup^e(t_1[A_n], t_2[A'_n]))\}\\
&= \{t \mid \exists t_1 \in \pi_{k(R)\cup I\cup A_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2))),\\
&\quad \exists t_2 \in \pi_{k(R)'\cup I\cup A'_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2))) \mid\\
&\quad (t[k(R)] = t_1[k(R)] = t_2[k(R)'])\wedge\\
&\quad (t[A_n] = t_1[A_n] = t_2[A_n] = \phi)\}\\
&\cup \{t \mid \exists t_{11}, ..t_{1p} \in \mu_{A_n}(\pi_{k(R)\cup I\cup A_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2)))),\\
&\quad \exists t_{21}, ..., t_{2q} \in \mu_{A'_n}(\pi_{k(R)'\cup I\cup A'_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2)))) \mid\\
&\quad (t[k(R)] = t_{11}[k(R)] = \cdots = t_{1p}[k(R)]\\
&\qquad = t_{21}[k(R)'] = \cdots = t_{2q}[k(R)'])\wedge\\
&\quad (t_{11}[I] = \cdots = t_{1p}[I] = t_{21}[I] = \cdots = t_{2q}[I])\wedge\\
&\quad (t[A_n] = \cup^e(\{t_{11}, ..., t_{1p}\}[Attr(A_n)], \{t_{21}, ..., t_{2q}\}[Attr(A'_n)]))\}\\
&= \pi_{k(R)\cup A_n}(\sigma_{A_n=A'_n=\phi}(r')) \cup\\
&\quad \pi_{k(R)\cup A_n}(\nu_{Attr(A_n)}(\cup^e(\mu_{A_n}(\pi_{k(R)\cup I\cup A_n}(Index(r'))), \mu_{A'_n}(\pi_{k(R)'\cup I\cup A'_n}(Index(r'))))))
\end{aligned}
$$

where $r' = \sigma_{k(R)=k(R)'}(r_1 \times r_2)$

Lemma 4 Let r_1 and r_2 be relations with relation scheme $R(A_1, ...A_n)$ and let $\mid m(R) \mid = s > 1$. Without loss of generality we can assume that $\{A_1, ...A_j\} = k(R)$, $(j = n - s)$ and $\{A_{j+1}, ...A_n\} = m(R)$. Then,

$$
\begin{aligned}
\cup^e(r_1 \times r_2) =& (r_1 - \pi_{A_1,...,A_n}(r')) \cup (r_2 - \pi_{A'_1,...,A'_n}(r')) \cup\\
& (X_{j+1}(r_1, r_2) \bowtie \cdots \bowtie X_n(r_1, r_2))
\end{aligned}
$$

where,

$$X_i(r_1, r_2) = \pi_{k(R)\cup A_i}(\nu_{Attr(A_i)}(\cup^e(\mu_{A_i}(\pi_{k(R)\cup I\cup A_i}(Index(r'))),$$
$$\mu_{A'_i}(\pi_{k(R)'\cup I\cup A'_i}(Index(r')))))) \cup \pi_{k(R)\cup A_i}(\sigma_{A_i=A'_i=\phi}(r')) \quad (j+1 < i < n)$$

and $r' = \sigma_{k(R)=k(R)'}(r_1 \times r_2)$ (R being the scheme of r_1 and r_2).

Proof:

The first two terms in the expression on the right hand side of the above equation are the same as those in the previous Lemma and their proofs are also identical. So we now have to prove that

$$X_{j+1}(r_1, r_2) \bowtie \cdots \bowtie X_n(r_1, r_2) = \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$$
$$\wedge (t[R_{j+1}] = \cup^e(t_1[R_{j+1}], t_2[R_{j+1}]))$$
$$\vdots$$
$$\wedge (t[R_n] = \cup^e(t_1[R_n], t_2[R_n]))\}$$

Let z denote the expression on the right hand side of the above equation

Induction Hypothesis Let Lemma 4 be true for all R s.t. $\mid m(R) \mid < s$.

Let $y = \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \mid$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$$
$$\wedge (t[R_{j+1}] = \cup^e(t_1[R_{j+1}], t_2[R_{j+1}]))$$
$$\wedge (t[R_{n-1}] = \cup^e(t_1[R_{n-1}], t_2[R_{n-1}]))\}$$

Therefore,

$$z = y \bowtie \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)])$$
$$\wedge (t[A_n] = \cup^e(t_1[A_n], t_2[A_n]))\}$$
$$= \cup^e(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2)) \bowtie$$
$$\cup^e(\pi_{Attr(R)-\{A_{j+1},\ldots,A_{n-1}\}}(r_1), \pi_{Attr(R)-\{A_{j+1},\ldots,A_{n-1}\}}(r_2))$$

Let R' be the scheme corresponding to $\pi_{Attr(R)-A_n}(r_1)$. $\mid m(R') \mid < s$. Hence, by the induction hypothesis,

$$\cup^e(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2)) = X_{j+1}(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2)) \bowtie \cdots \bowtie$$
$$X_{n-1}(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2))$$

Now,

$$X_{j+1}(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2)) = \pi_{k(R')\cup A_{j+1}}(\nu_{Attr(A_{j+1})}$$
$$(\cup^e(\mu_{A_{j+1}}(\pi_{k(R')\cup I\cup A_{j+1}}(Index(r'')))),$$
$$\mu_{A'_{j+1}}(\pi_{k(R')'\cup I\cup A'_{j+1}}(Index(r'')))))$$
$$\cup\ \pi_{k(R')\cup A_{j+1}}(\sigma_{A_{j+1}=A'_{j+1}=\phi}(r''))$$

where, $r'' = \sigma_{k(R')=k(R')'}(\pi_{Attr(R)-A_n}(r_1) \times \pi_{Attr(R)-A_n}(r_2))$

Now,

$$\pi_{k(R')\cup I\cup A_{j+1}}(Index(\sigma_{k(R')=k(R')'}(\pi_{Attr(R)-A_n}(r_1) \times \pi_{Attr(R)-A_n}(r_2))))$$
$$= \pi_{k(R)\cup I\cup A_{j+1}}(\pi_{Attr(R)\cup Attr(R)'\cup I-A_n-A'_n}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2))))$$

since $A_n \in m(R), k(R') = k(R)$ and $k(R')' = k(R)'$

$$= \pi_{k(R)\cup I\cup A_{j+1}}(Index(\sigma_{k(R)=k(R)'}(r_1 \times r_2)))$$

and $\pi_{k(R')\cup A_{j+1}}(\sigma_{A_{j+1}=A'_{j+1}=\phi}(r'')) = \pi_{k(R)\cup A_{j+1}}(\sigma_{A_{j+1}=A'_{j+1}=\phi}(\sigma_{k(R)=k(R)'}(r'))$

Thus we have $X_{j+1}(\pi_{Attr(R)-A_n}(r_1), \pi_{Attr(R)-A_n}(r_2)) = X_{j+1}(r_1, r_2)$.
We get similar results for $X_{j+2}(r_1, r_2), ..., X_{n-1}(r_1, r_2)$
Now, $\cup^e(\pi_{Attr(R)-\{A_{j+1},...,A_{n-1}\}}(r_1), \pi_{Attr(R)-\{A_{j+1},...,A_{n-1}\}}(r_2)) = X_n(r_1, r_2)$
since, $m(R^n)$, where R^n is the scheme corresponding to the relations in the union expression, $= \{A_n\}$, $\mid m(R^n) \mid = 1$ and hence by Lemma 3 the above equation is true.

Therefore, $z = X_{j+1}(r_1, r_2) \bowtie, ..., \bowtie X_n(r_1, r_2)$
Hence, Lemma 4 is true.

Theorem 2 If r_1 and r_2 are two relations with relation scheme R, then $\cup^e(r_1, r_2)$ can be expressed in terms of $\sigma, \times, \pi, \nu, \mu, \cup$ and $-$.

Outline of Proof The proof for this Theorem is by induction on the depth of the relation scheme R, using Lemma 3 and Lemma 4. (Note that $\bowtie$ can be expressed in terms of $\times$ and σ .)

In the case of the extended difference operator, $-^e$, although it may seem that the equivalent expression in the non-recursive algebra would be almost identical to the

expression in Lemma 4, with ∪ replaced by −, there is in fact a slight difference. The following example shows this difference. Figure 31 is the result of applying the extended difference operator to the relations in Figures 29 and 30 while Figure 32 shows the result of applying the expression in Lemma 3 with ∪ replaced by − and without the second term in the expression.

m_1

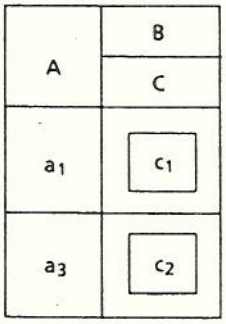

Fig. 29

m_2

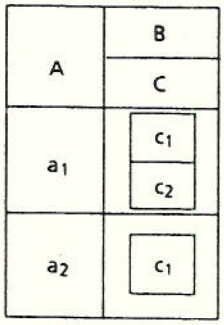

Fig. 30

$-^e(m_1, m_2)$

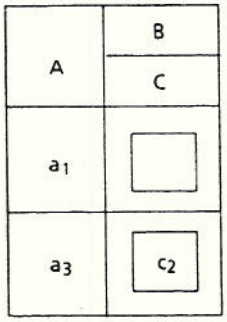

Fig. 31

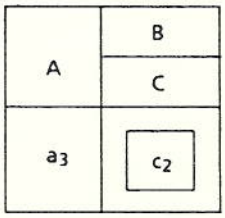

Fig. 32

When the relations m_1 and m_2 are unnested on the attribute B and the difference is taken, the tuple $(a_1\ c_1)$ disappears while when the extended difference is applied, the result contains the tuple $(a_1\ \{\ \})$.

Lemma 5 Let r_1 and r_2 be two relations with relation scheme $R(A_1, ..., A_n)$. Let $k(R) = \{A_1, ..., A_j\}$ and let $m(R) = \{A_{j+1}, ..., A_n\}$. Then,

$$-^e(r_1, r_2) = (r_1 - \pi_{A_1,...,A_n}(r')) \cup (((\pi_{k(R)}(r_1') - \pi_{k(R)}(Q_{j+1}(r_1, r_2))) \times E(A_{j+1})) \cup Q_{j+1}(r_1, r_2)) \bowtie$$
$$\cdots \bowtie (((\pi_{k(R)}(r_1') - \pi_{k(R)}(Q_{j+1}(r_1, r_2))) \times E(A_{j+1})) \cup Q_{j+1}(r_1, r_2))$$

where,

$$Q_i(r_1, r_2) = \pi_{k(R)\cup A_i}(\nu_{Attr(A_i)}(-^e(\mu_{A_i}(\pi_{k(R)\cup I\cup A_i}(Index(r'))),$$
$$\mu_{A_i'}(\pi_{k(R)'\cup I\cup A_i'}(Index(r')))))) \quad (j+1 < i < n)$$

$r' = \sigma_{k(R)=k(R')}(r_1 \times r_2)$, $E(A_i)$ is a relation with a single attribute A_i and a single tuple whose value $= \phi$

and $r_1' = r_1 - \sigma_{Attr(R)=Attr(R)'}(r_1 \times r_2)$

Proof

Case 1 $m(R) = \{A_n\}$ imp $\mid m(R) \mid = 1$

$$-^e(r_1, r_2) = \{t \mid ((t \in r_1) \wedge (\forall t_2 \in r_2, t_2[k(R)] \neq t[k(R)]))$$
$$\vee (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)]) \wedge (t_1 \neq t_2)$$
$$\wedge (t[R_n] = -^e(t_1[R_n], t_2[R_n])))\}$$

Now, $\{t \mid t \in r_1 \mid \forall t_2 \in r_2, t[k(R)] \neq t_2[k(R)]\}$
$= r_1 - \pi_{A_1,\ldots,A_n}(r')$ where $r' = \sigma_{k(R)=k(R)'}(r_1 \times r_2)$
Therefore,

$$-^e(r_1, r_2) = (r_1 - \pi_{A_1,\ldots,A_n}(r')) \cup \{t \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)]) \wedge (t_1 \neq t_2)$$
$$\wedge (t[R_n] = -^e(t_1[R_n], t_2[R_n])))\}$$
$$= \{t \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)]) \wedge (t_1 \neq t_2)$$
$$\wedge ((t[R_n] = -^e(t_1[R_n], t_2[R_n]) \neq \phi) \vee (t[A_n] = t_1[A_n] = \phi))\}$$
$$\cup \{t \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 \mid$$
$$(t[k(R)] = t_1[k(R)] = t_2[k(R)]) \wedge (t_1 \neq t_2)$$
$$\wedge (t[R_n] = -^e(t_1[R_n], t_2[R_n]) = \phi))\}$$

The first term in the last equation's r.h.s. can be proved to be $Q_n(r_1, r_2)$ and the second can be seen to equal to $((\pi_{k(R)}(r_1') - \pi_{k(R)}(Q_n(r_1, r_2))) \times E(A_n))$.

The proof for the case when $\mid m(R) \mid > 1$ is by induction and similar to the proof for Lemma 4. Finally, it can be proved by induction on the depth of R that the extended difference operator is expressible in terms of the non-recursive operators.

Claim: Any query expressed in the algebra of Thomas and Fischer can be expressed in the recursive algebra with either the same number or fewer operators than in the former.

Proof:

We consider only the basic operators $\sigma, \pi, \times, \nu, \mu, \cup$, and $-$, since the other operators are all expressible in terms of the basic ones.

Now,

(1) $\sigma_c(r) = \sigma'(r_c)$

(2) $\pi_A(r) = \pi'(A(r))$

(3) $r_1 \times r_2 = \times'(r_1, r_2)$

(4) $\nu_{A \to A'}(r) = \nu'((A \to A')r)$

(5) $\mu_A(r) = \mu'(A(r))$

(6) $r_1 \cup r_2 = \cup'(r_1, r_2)$

(7) $r_1 - r_2 = -'(r_1, r_2)$

Let $\alpha = \alpha_1 ... \alpha_m$ be an expression in the algebra of Thomas and Fischer. Each α_i is a basic operator.

(1) If we replace each α_i by the corresponding α'_i given in the equation set above we get an equivalent expression α^r in the recursive algebra such that $\alpha^r = \alpha'_1 ... \alpha'_m$ which has the same number of operators as α.

(2) If there exists a string of operators β in α such that $\alpha = \alpha_1 ... \alpha_k \beta ... \alpha_m$ and $\beta = P_n(r)$ where P_n is defined in the previous proof, then β, which is of length > 1, can be replaced by $\sigma'(r_c(R_{1c_1}L_1, ..., R_{nc_n}L_n))$. This gives us an expression that has fewer operators than the original one.

(3) The other operators can be replaced in a similar fashion as in step 2. Step 2 is thus applied repeatedly until there are no more strings of the type β that can be replaced. Then we apply step 1 to all the remaining operators in α that have not been replaced by operators of the recursive algebra.

(4) The result is an expression with the same or fewer operators than α.

5. Summary and Conclusions

The nested relational model, like the traditional (flat) relational model, is one that is simple and has a strong theoretical basis, but provides a much better way of representing complex data than the relational model. Most query languages for nested relations, however, do not preserve the structure of complex objects while accessing the database. Complex objects often have to be broken down into their constituent parts while being queried upon and then rebuilt from these parts. Not only does this cause a lot of inconvenience to the users but it is also expensive in terms of the time that it takes to perform the restructuring operations.

In this paper, we have proposed a recursive query language that can navigate through the complex structure of the nested relations without always restructuring them thus reducing the number of restructuring operations. We have also given a sketch of a proof to show that this language is as powerful as one of the earlier query languages for nested relations that was developed by Thomas and Fischer and that queries in this algebra can always be expressed using fewer or the same number of operators as an equivalent query written in the non-recursive algebra.

Further research can be done on this language in several areas including the development of query optimization principles. In the relational algebra, query optimization is based on the commutativity property of operators. These principles could not be extended to the nested relational algebra since most expressions in this algebra involve the nest and unnest operators which do not commute either with each other or with the other operators. In the recursive algebra, nest and unnest are used only when restructuring is really necessary, *i.e.*, they are not needed for accessing the interior levels in a nested relation and so most queries can be formulated by expressions that do not involve these two operators. Hence, queries in the recursive algebra are not only more succinct and more efficient as compared to those in the non-recursive nested algebra, but may also be efficiently optimizable using techniques that are similar to those of the relational algebra.

Acknowledgements

I would like to thank José Blakeley, Marc Gyssens, Sid Kitchel, Nancy Martin, and Dirk Van Gucht for their helpful suggestions and ideas.

References

1. Abiteboul, S., and Bidoit, N., "Non First Normal Form Relations to Represent Hierarchically Organized Data," *Proc. Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1984, pp. 191-200.

2. Bancilhon, F., Richard, P., and Scholl, M., "On Line Processing of Compacted Relations," *Proc. 8th VLDB*, Mexico City, 1982, pp. 21-37.

3. Codd, E.F., "A Relational Model for Large Shared Data Banks, " *Communications ACM* Vol. 6, No. 13, June 1970, pp. 377-387.

4. Dadam, P., Kuespert, F., Andersen, F., Blanken, H., Erbe, R., Guenauer, J., Lum, V., Pistor, P., and Walch,G., "A DBMS Prototype to Support Extended NF2 Relations: An Integrated View on Flat Tables and Hierarchies," *Proc. Annual SIGMOD Conf.*, Austin, 1986, pp. 356-366.

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

6. Deshpande, V., and Larson, P.A., "An Algebra for Nested Relations," *Tech. Report, CS-87-65*, 1987, University of Waterloo.

7. Deshpande, A., and Van Gucht, D., "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.

8. Gyssens, M. and Van Gucht, D., "The Powerset Algebra as a Result of Adding Programming Constructs to the Nested Relational Algebra," *Proceedings of SIGMOD Conf.*, Chicago, IL, June 1988

9. Jaeshke, G., and Schek, H.J., "Remarks on the Algebra on Non-First Normal Form Relations," *Proc. 1st PODS,* Los Angeles, 1982, pp. 124-138.

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

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

12. Ozsoyoglu, G., Ozsoyoglu, Z.M., and Matos, V., "Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions," *ACM Transactions on Database Systems*, Vol. 12, No. 4, December 1987, pp. 566-592.

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

14. Pistor, P., and Andersen, F., "Designing a generalized NF2 model with an SQL-Type Language Interface," *Proc. 12th VLDB,* Kyoto, Japan, 1986, pp. 278-288.

15. Pistor, P., and Traunmueller, R., "A Database Language for Sets, Lists and Tables," *Information Systems*, Vol 11, No. 4, 1986, pp. 323-336.

16. Roth, M.A., Korth, H.F., and Batory, D.S., "SQL/NF: A Query Language for ¬1NF Relational Databases," *Information Systems*, Vol 12, No. 1, 1987, pp. 99-114.

17. Roth, M.A., Korth, H.F., and Silberschatz, A., "Theory of Non-First-Normal-Form Relational Databases," *Tech. Report TR-84-36 (Revised January 1986)*, University of Texas at Austin, 1984.

18. Schek, H.J., and Scholl, M.H., "The Relational Model with Relation-Valued Attributes," *Information Systems*, Vol. 11, No. 2, 1986, pp. 137-147.

19. Scholl, M.H., Paul, H.B., and Schek, H.J., "Supporting Flat Relations by a Nested Relational Kernel," *Proc. 13th VLDB*, London, 1987.

20. Thomas, S.J., and Fischer, P.C., "Nested Relational Structures," Advances in Computing Research III, The Theory of Databases, P.C. Kanellakis, ed.,JAIpress, 1986, pp. 269-307.

21. Tsichritzis, D.,and Lochovsky, F.H. "Hierarchical Data-Base Management: A Survey," *ACM Computing Surveys* Vol. 8, No. 1, March 1976, pp. 105-123.

22. Van Gucht, D., "Theory of Unnormalized Relational Structures," Ph.D. Dissertation, Vanderbilt University, 1985.

23. Van Gucht, D., "On the Expressive Power of the Extended Relational Algebra for the Unnormalized Relational Model," *Proc. 6th PODS*, San Diego, CA, March 1987, pp. 302-312.

24. Van Gucht, D., and Fischer, P.C., "Multilevel Nested Relational Structures," *Journal of Computer and System Sciences*, Vol. 36, No. 1, February 1988, pp. 77-105