Inform. Systems Vol. 11, No. 2, pp. 137–147, 1986
Printed in Great Britain.

0306–4379/86 $3.00 + .00
© 1986 Pergamon Press Ltd.

THE RELATIONAL MODEL WITH RELATION-VALUED ATTRIBUTES

H.-J. SCHEK and M. H. SCHOLL
Technical University of Darmstadt, Alexanderstrasse 24, D-6100 Darmstadt, West Germany

(*Received* 17 *August* 1984; *in revised form* 18 *June* 1985)

Abstract—Motivated by new applications of database systems we introduce relations which may have relation-valued attributes and propose a related algebra. Formal definitions for this extended relational model can be given by applying usual notions recursively. The main problem considered in this paper is the formal definition of an appropriate relational algebra for these non-first-normal-form relations. We allow the application of the basic relational operators to any relation-valued attribute within a relation. This leads to a (hierarchically) nested relational algebra.

1. INTRODUCTION

The First Normal Form (1NF) condition of the relational model of data[1] is widely accepted as a reasonable restriction for the relations to be considered further. In [2, p. 239] we find: "The only hard requirement is that relations be in at least first normal form". Nevertheless we have practical and theoretical reasons for investigations on relations which may have relations as attribute values which, in turn, may have relations as attributes values and so on.

On the practical side, we try to get a better solid basis for new applications of data base management systems (DBMS). Examples are textual data, engineering data or images; that is, objects which usually have not been considered as data to be stored in currently available DBMS. In this situation, preprocessor solutions are the usual way to overcome the difficulties. A DBMS is then used as a byte string storage manager with the responsibility of the preprocessor to take care of the interpretation of the byte string by imposing a data structure on it. The consequence is that the preprocessor becomes responsible for many primitive functions such as updating, indexing, or integrity control functions which are normally meant to be managed by the DBMS.

We believe that some of the bottlenecks can be avoided by dropping the 1NF and allowing relation-valued attributes. User or application specific data structures and operations, e.g. retrieving elements of the *k*-th row of a huge matrix should then be better supported by this model than by the naive storage of the large matrix as a long byte string. Furthermore, already for conventional applications of DBMS we observe that repeating groups or sets occur often in the description of real world objects. Disallowing those as attributes (in the sense of the Entity-Relationship-Model) leads to an artificial separation into several (entity) relations. Constructs such as weak entities[3] or referential integrity contraints must be introduced to maintain integrity. From a theoretical point of view we have been motivated by the question whether we are able to keep the main formalism: e.g. according to [4] for relations with relation-valued attributes in such a way that the 1NF case is a degeneration.

Several authors already investigated "Non-First-Normal-Form" relations, shortly called NF^2 relations in [5, 6]. Aspects mainly of the design theory with unnormalized relations and transformations of dependencies are contained in [7–11].

A discussion on the role of the new algebra operators nest and unnest in their interactions with the usual ones (e.g. whether a projection and a nest on a relation can be computed, or whether a specific NF^2 relation can be reached by a sequence of nests and unnests starting with a 1NF relation) is contained in [12, 9]. An extension of the selection operation for relation-valued attributes is contained in [13].

A general formal description of hierarchical data structures has been given recently by [14]. The application of hierarchically structured data for the user interface called "forms" and the conversion problem between data models is described in [15–19]. The introduction of so-called complex objects in [20] for engineering applications is a first step into the direction of allowing unnormalized relations. The effect of "quotient relations"[21] can also be described in terms of the NF^2 model. A proposal for generally allowing recursive data models is described in [22].

An interesting connection exists also to the "compacted relations"[23] for hardware search facilities. The transformation from 1NF relational expressions at a user interface into equivalent expressions for the internal compacted relations can be described by the NF^2 relational algebra. In fact, our approach is to describe all data at the three levels (external, conceptual and internal) according to [24], especially including internal data structures like access paths as NF^2 relations. As we then have a single formal model for the conceptual and internal schema we get the mappings between these by NF^2 relational algebra expressions. This aspect is described in [6].

In this paper, we restrict ourselves to a foundation of the NF^2 relational model. We give a precise definition of the NF^2 relational data structure and of the related NF^2 relational algebra. As far as known to the authors, no satisfying definition of a complete NF^2 relational algebra has been proposed until now.

The report is structured in the following way: We give a formal description of an NF^2 relation and its several components in section 2 where we extend the usual notions by introducing recursion to reveal the nested structure. In section 3, we propose an NF^2 relational algebra, for which we give recursive definitions too. First, we use our formalism to define the usual 1NF algebra precisely and in a second step we extend selection and projection with the objective to allow algebra expressions also within the selection formula or within the projection list in a nested manner. Finally, we show that the nested selection can be expressed by a suitable projection-selection expression. This serves as a first example for equivalent NF^2 algebra expressions.

2. RELATIONS WITH RELATION-VALUED ATTRIBUTES

We try to preserve the usual notions of schema, value and attributes of a relation, etc. The formal definitions which follow will contain the 1NF case as a degeneration.

At a first glance the reader might be astonished at the formalism introduced for more or less common facts. But in case of NF^2 relations, we cannot dispense with formal definitions of basic concepts like schema, value, attributes and tuples of relations, because otherwise we could not define the effects of relational algebra operators formally.

2.1 *The NF^2 relational data structure*

In the 1NF case we assume that δ sets D_i, $i = 1, 2, \ldots, \delta$ are given which serve as "basic domains". The elements from D_i will not be regarded as decomposable by the DBMS. For our purpose of defining a relational algebra it is sufficient to introduce one basic domain $D := D_1 \cup \cdots \cup D_\delta$. (This setting simplifies the discussion without loosing generality. The restriction of attribute values to certain subsets of D (e.g. D_1) can be regarded as a special kind of integrity constraints which are not of interest in this paper.) We further assume a given set **NAME** the elements of which will be used for naming purposes.

We denote with $\mathbf{P}(S)$ the powerset of a set S and with $S_1 \times S_2$ the Cartesian product of two sets S_1 and S_2. $\varnothing$ is the empty set. For the elements of a Cartesian product we write $\langle\cdots\rangle$, that is, for $s \in S_1 \times S_2$ we can write $s = \langle s_1, s_2\rangle$, with $s_1 \in S_1$ and $s_2 \in S_2$. ($\exists$) and ($\forall$) denote the existential and universal quantifiers respectively.

The following definitions (1, 2, 3) are preparations for defining NF^2 relations.

Definition 1 *"Complex Domains C"*.
The set **C** of "complex domains" is defined by
(1) $D \in \mathbf{C}$
(2) $C_1, C_2, \ldots, C_k \in \mathbf{C}$
$\Rightarrow \mathbf{P}(C_1 \times C_2 \times \cdots \times C_k) \in \mathbf{C}$
(3) nothing else is in **C** □

Definition 1 simply allows powersets of Cartesian products of the (basic or complex) domains to be also a complex domain.

Definition 2 *"Schema S and Description B"*.
The set of allowed schemata **S** and descriptions **B** is defined by
(1) $N \in \mathbf{NAME} \Rightarrow \langle N, \varnothing\rangle \in \mathbf{B}$
(2) $b_i = \langle N_i, S_i\rangle \in \mathbf{B}$, $i = 1, \ldots, k$, $N_i \neq N_j$ for $i \neq j \Rightarrow \{b_1, b_2, \ldots, b_k\} \in \mathbf{S}$
(3) $N \in \mathbf{NAME} \wedge S \in \mathbf{S} \Rightarrow \langle N, S\rangle \in \mathbf{B}$ □

Obviously, the intention of definition 2 is to have a pair as a description consisting of a name as first component and a schema as second. On the other hand, a schema is a *set* of descriptions, especially a schema might be the empty set. Further, the names which appear in the descriptions b_i of a single schema must be unique.

As we assume one basic domain D only (see above), the domain of a given description is uniquely determined by its schema, as we see from the following definition.

Definition 3 *"Description-Related Domain"*.
We define a function dom: $\mathbf{B} \rightarrow \mathbf{C}$ by

$$\langle N, S\rangle \in \mathbf{B} \Rightarrow$$
$$\mathrm{dom}(\langle N, S\rangle) = \begin{cases} D & \text{for } S = \varnothing \\ \mathbf{P}(\underset{y\in S}{\times}\ \mathrm{dom}(y)) & \text{else} \end{cases} \qquad \square$$

Obviously, the values of this function are in **C**.

Definition 4 "NF^2 *Relation*". Given $b = \langle N, S\rangle \in \mathbf{B}$ a pair $R = \langle b, v\rangle$ is an NF^2 relation, iff $S \neq \varnothing$ and $v \in \mathrm{dom}(b)$. □

In the usual terminology v is called "instance" or "value" of the schema defined by b. We do not consider further restrictions for v, e.g. satisfaction of integrity constraints, throughout this paper because this aspect is not relevant for the definitions given. Furtheron we use the following notational conventions:

For the two components of a relation R we use
(1) des(R) for the first component of R
(2) val(R) for the second component of R

For the two components of a description $b \in \mathbf{B}$ we use
(3) nam(b) for the first component of b
(4) sch(b) for the second component of b

Therefore, if $b = \langle N, S\rangle = \mathrm{des}(R)$ is the description of a given relation R we get by
nam(des(R)) the name (N)
sch(des(R)) the schema (S)

of the given relation R. As we will often need the schema of a relation to describe its transformation by algebraic operations to be defined in the next chapter, we extend the function sch to be directly applicable to relations by the following definition:

(5) $\text{sch}(R) := \text{sch}(\text{des}(R))$

An element $y \in \text{sch}(R)$ is called an "attribute description" and its first component $\text{nam}(y)$ is called an "attribute" of relation R. This corresponds to the usual terminology. As the schema of a relation is the set of attribute descriptions and not only the set of attributes (that is, names) as in the flat relational model we introduce $\text{attr}(R)$ to be the set of attributes according to

(6) $\text{attr}(R) := \text{attr}(\text{des}(R)) = \{N \mid (\exists)\ y \in \text{sch}(R) \wedge N = \text{nam}(y)\}$

Definition 2 requires that the elements of a single schema have different names (that is, for any given schema $S \in \mathbf{S}$ and $b_1 \in S$, $b_2 \in S$, $b_1 \neq b_2$ we have $\text{nam}(b_1) \neq \text{nam}(b_2)$). Therefore we use the function $d_R\text{:attr}(R) \rightarrow \text{sch}(R)$ which is inverse to nam according to

(7) $A \in \text{attr}(R) \Rightarrow (\exists)\ y \in \text{sch}(R) \wedge d_R(A) = y \wedge \text{nam}(y) = A$

In the following we will omit the subscript "R" of the function d_R when the meaning is clear from the context. An element of a relation value is called a tuple; that is,

(8) R is a relation $\Rightarrow t \in \text{val}(R)$ is a tuple of R

We know that $\text{val}(R) \in \mathbf{P}(\underset{y \in \text{sch}(R)}{\times} \text{dom}(y))$.

Therefore a tuple has as many components as the cardinality of $\text{sch}(R)$. If $\text{sch}(R) = \{y_1, y_2, \ldots, y_k\}$ and if the order is fixed for the moment, we can state $t = \langle t_{y1}, t_{y2}, \ldots, t_{yk}\rangle$ with $t_{yi} \in \text{dom}(y_i)$. As the following lemma shows, we can adopt the set-of-mapping notion of a relation:

Lemma 1:
A tuple t of a relation R is a mapping of the form
$t\text{: attr}(R) \rightarrow \bigcup_{y \in \text{sch}(R)} \text{dom}(y)$

Proof: We prove this by an explicit assignment: For $A_i \in \text{attr}(R)$ with $d(A_i) = y_i \in \text{sch}(R)$ we set $t(A_i) = t_{yi}$. As $t_{yi} \in \text{dom}(y_i)$ the function t has the desired property. □

Definition 6 *"Attribute Value"*.
Let R be an (NF^2) relation, $t \in \text{val}(R)$ and A an attribute. The value of the function $t(A)$ is called "attribute value" of A.
For a set of attributes $A = \{A_1, \ldots, A_n\} \subseteq \text{attr}(R)$, $t \in \text{val}(R)$, we also apply the usual shorthand notation

$$t(A) := t(A_1, A_2, \ldots, A_n) := \langle t(A_1), t(A_2), \ldots, t(A_n)\rangle \quad \square$$

Lemma 2:
Under the assumptions of definition 6 the pair $\langle d(A), t(A)\rangle$ is a relation, iff $\text{sch}(d(A)) \neq \emptyset$

Proof: Obviously, the first component $d(A)$ is in $\mathbf{B}$ since $A \in \text{attr}(R)$. For the second component we have, according to lemma 1, $t(A) \in \text{dom}(d(A))$ which is in $\mathbf{C}$ (see definition 3). □

Lemma 2 indicates that an attribute value together with its attribute description may again be a (1NF or NF^2) relation supposed it has a non-empty schema. If the schema of an attribute description is empty, the attribute value is from the atomic domain D (see definition 3). Therefore we can characterize a 1NF relation as a special case of a relation.

Lemma 3: A relation R is a 1NF relation
$\Leftrightarrow (\forall)\ y \in \text{sch}(R)\text{: sch}(y) = \emptyset$
A relation R is a "real" NF^2 relation
$\Leftrightarrow (\exists)\ y \in \text{sch}(R) \wedge \text{sch}(y) \neq \emptyset$ □

The following consequences can be drawn from these definitions.

1. We have introduced the notion of "attribute description" which was not necessary in the flat model (where the description degenerates to $\langle\text{attribute}, \emptyset\rangle$). On the other side, we use this format also for the description of a relation in the form $\langle$relname, schema$\rangle$, where now the second component schema is important, also for the 1NF case. The notion of the attribute value in a tuple is not changed.
2. As an attribute value together with its attribute description forms a relation which, in turn, may have attributes which are relation-valued we are able to apply the definitions and notation introduced so far in a nested manner. Especially, if $r = \langle d(A), t(A)\rangle$ is a relation consisting of the description $d(A)$ and the value $t(A)$ of an attribute A of a relation R and if $a \in \text{attr}(r)$ is an attribute and $\tau \in \text{val}(r)$ is a tuple of r we have $\tau(a)$ as an attribute value from $\text{dom}(d(a))$.
3. The operations of the NF^2 algebra to be defined in the next chapter may be applied in a nested manner; that is, to a given relation R but also to relations which are constructed from its relation-valued attributes.

2.2 *Representations of NF^2 relations*

2.2.1 *Linear form of the schema.* We extend the usual linear form $R(A_1, A_2, \ldots, A_n)$ for a schema of a 1NF relation R with attributes $A_1, \ldots, A_n$ by repeating the linear form for RV-attributes. In the example

$$R\ (A_1\ (A_{11}, A_{12}), A_2, A_3\ (A_{21}, A_{32}\ (A_{321}, A_{322}), A_{33}))$$

we have A_2 as AV-attribute and two RV-attributes A_1 and A_3.

2.2.2 *Tree representation of the schema.* The linear form of a schema may also be represented by a tree with R as root node and the AV-attributes as leaves. Non-leaf nodes different from the root represent *RV*-attributes. In the example above we get

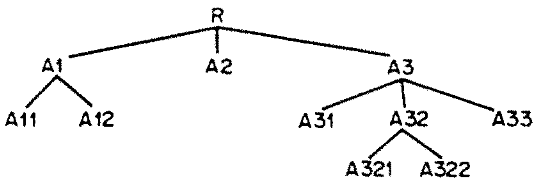

Fig. 1. Tree representation of *R*.

2.2.3 *Tabular representation of the schema.* In order to show the schema and the value of an NF^2 relation together we may also apply a tabular representation as shown by the sample relation DEPT in the next section which will be referred to by examples in the following chapter.

2.2.4 *Sample relation and its different representations.* We consider a relation DEPT containing information about departments with their administrational and technical staff including sets of courses the technical employees attended. The schema of the DEPT relation in its linear form looks like the following:

$$\text{DEPT}\ (D,\ \text{DN},\ \text{AE},\ (\text{AN},\ \text{AJD}),\ \text{TE}\ (\text{TN},\ \text{TJD},\ C\ (\text{CN},\ Y)))$$

This schema can be represented by the tree of Fig. 2:

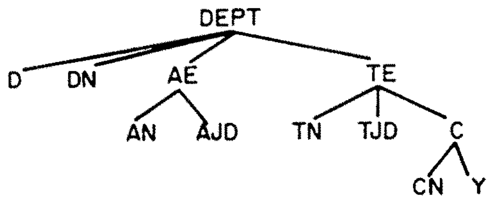

Fig. 2. Tree representation of the DEPT relation.

The tabular representation of the schema and a sample value of DEPT is shown in Fig. 3.

DEPT							
D	DN	AE		TE			
		AN	AJD	TN	TJD	C	
						CN	Y
1	RESEARCH	121	LIBRARY	511	SOFTWARE ENGINEERING	1	75
		122	SECRETARY			2	76
		130	TRANSLATION			5	79
		140	PATENTS	552	BASIC RESEARCH	1	82
		169	SECRETARY			2	79
				678	PLANNING	2	76
						4	82
2	DEVELOPMENT	119	SECRETARY	650	DESIGN	1	75
		125	TRANSLATION			2	77
		135	PROCUREMENT	780	MAINTENANCE	3	82
				981	PLANNING	2	81
						3	82

Fig. 3. Tabular representation of a sample DEPT relation.

2.3 *Example applications of the formal definitions*

The relation DEPT of Fig. 3 has the description (according to (1))

$$\text{des(DEPT)} = \langle \text{DEPT}N, \{\langle D, \varnothing\rangle, \langle \text{DN}, \varnothing\rangle, \langle \text{AE}, S_1\rangle, \langle \text{TE}, S_2\rangle\}\rangle \text{ with}$$

$$S_1 = \{\langle \text{AN}, \varnothing\rangle, \langle \text{AJD}, \varnothing\rangle\}$$

$$S_2 = \{\langle \text{TN}, \varnothing\rangle, \langle \text{TJD}, \varnothing\rangle, \langle C, \{\langle \text{CN}, \varnothing\rangle, \langle Y, \varnothing\rangle\}\rangle\}$$

The attribute description of attribute AE in DEPT is according to (7)

$$d(\text{AE}) = \langle \text{AE}, \{\langle \text{AN}, \varnothing\rangle, \langle \text{AJD}, \varnothing\rangle\}\rangle$$

The schema part of this attribute is $\text{sch}(d(\text{AE})) = \{\langle \text{AN}, \varnothing\rangle, \langle \text{AJD}, \varnothing\rangle\}$ which is not empty. Therefore AE is a relation-valued attribute (see definition 4). A sample value of this attribute is (see definition 6)

$$t(AE) = \{\langle 119, \text{SECRETARY}\rangle \\ \langle 125, \text{TRANSACTION}\rangle \\ \langle 135, \text{PROCUREMENT}\rangle\}$$

which appears in the "second" tuple $t \in \text{val(DEPT)}$.

The relation (see lemma 2) $\text{AE}R := \langle d(\text{AE}), t(\text{AE})\rangle$ is a 1NF relation (see lemma 3) since

$$\text{attr}(\text{AE}R) = \{\text{AN}, \text{AJD}\} \quad \text{and}$$

$$\text{sch}(d(\text{AN})) = \text{sch}(d(\text{AJD})) = \phi$$

The DEPT relation is a real NF^2 relation. For this "outer" relation there is one description and only one value at any point in time. Therefore we could have used the name nam(des(DEPT)) = DEPT*N* to identify the relation. This, however, can not be done for "subrelations" such as AE*R*. The relation is not uniquely identified by the name AE but rather by the name AE *and* the tuple $t \in \text{val(DEPT)}$ under consideration. AE only identifies the description d(AE).

We construct the domain of AE according to definition 3 as

$$\text{dom}(\langle \text{AE}, \{\langle \text{AN}, \varnothing\rangle, \langle \text{AJD}, \varnothing\rangle\}\rangle) \\ = \text{P}(\text{dom}(\langle \text{AN}, \varnothing\rangle) \times \text{dom}(\langle \text{AJD}, \varnothing\rangle)) \\ = \text{P}(D \times D)$$

3. THE NF^2 RELATIONAL ALGEBRA

3.1 *General remarks*

Since NF^2 relations are sets, the set operations union ($\cup$) and difference ($-$) are applicable without any changes to the NF^2 case. The selection (σ_F) will select *whole* tuples based on the predicate F to be applied on a tuple. The projection (π) will determine *specific* components of all tuples of a relation and eliminate duplicates. In that aspect no change is introduced compared to the 1NF case. Also the definition of the Cartesian product (X) is unchanged.

The only new operations to be introduced are "nest" (ν) and "unnest" (μ). They are used to transform between 1NF and NF^2 relations and vice versa. Therefore, as the 1NF relational algebra has 5 independent operations ($\cup$, $-$, X, σ, π) we will now have 7 operations ($\cup$, $-$, X, σ, π, ν, μ) in the NF^2 case. This is not exciting and, in fact, would not allow an effective manipulation of NF^2 relations.

However, since attribute values may be relations whose attribute values again may be relations, etc., a *nested* application of the basic operations mentioned above leads to a high level of expressiveness. In order to formally define nested expressions we will first define the seven "building blocks" in such a way that we are able to extend some of them to become nested in a second step.

Generally, we use recursive definitions for the algebraic operators. Any operation applied to a (pair of) relation(s) yields a relation, therefore we can build algebraic *expressions* by recursively applying algebraic *operations*. Usually these kind of expressions are called "nested" expressions. However, in our case, relation-valued attributes allow another type of recursive application of the algebra (now within the square brackets "[. . .]" as opposed to the parentheses "(. . .)" as usual). In this paper we use the terminology of *expressions* versus *operations* for the first kind of recursion and *nested* algebra for the second one.

It is also important to inspect the structural effects of our algebra more carefully. Usually, in the 1NF case, the schema transformations caused by several algebraic operators are not described explicitly.

3.2 *Schema preserving operations*

In the 1NF case there are three out of the five basic operations that do not affect the relation schema: the two binary set operations union and difference and the unary relational selection.

3.2.1 *Set operations.* As in the 1NF case set operations on NF^2 relations $R1$ and $R2$ are only defined if sch($R1$) = sch($R2$).

Definition 7 *"set operations".* If sch($R1$) = sch($R2$) then

(1) sch($R1 \cup R2$) := sch($R1 - R2$) := sch($R1 \cap R2$) := sch($R1$)

(2) (a) val($R1 \cup R2$) := val($R1$) $\cup$ val($R2$)
(b) val($R1 - R2$) := val($R1$) $-$ val($R2$)
(c) val($R1 \cap R2$) := val($R1$) $\cap$ val($R2$)
(=val($R1 - (R1 - R2)$)) □

3.2.2 *Selection.* We give a modular definition which corresponds to the usual selection. The reason for these explicit definitions will be seen when we extend the definitions to the nested selection (see 3.4.1).

Definition 8 *"selection atom".*
A selection atom (SA) has one of the following two forms:

Type SA1: attribute cop constant
Type SA2: attribute1 cop attribute2

cop $\in \{<, \leq, >, \geq, =, \neq, \subset, \subseteq, \supset, \supseteq, \in, \ni\}$ □

Notice that in the 1NF case the set comparison operators would be omitted.

Definition 9 *"selection formula".* A selection formula (SF) is a logical combination of selection atoms:

(1) Each SA is a SF
(2) S is a SF $\Rightarrow$ (*not* F) is a SF
(3) F_1, F_2 are SF $\Rightarrow$ ($F_1 \wedge F_2$), ($F_1 \vee F_2$) are SF
(4) nothing else is a SF □

Now we can give a definition of the selection operation $\sigma[F](R)$.

Definition 10 *"selection".*

(1) sch($\sigma[F](R)$) := sch(R)
(2) The value of a selection is determined according to 4 cases, depending on the type of F:
 (a) If F is a selection-atom, then two subcases are regarded:
  (i) F is SA1-type; that is, $F = A$ cop C if $A \in$ attr(R) and $C \in$ dom($d(A)$), then
  val($\sigma[F](R)$) := $\{t \mid t \in$ val(R) $\wedge\ t(A)$ cop $C\}$
  (ii) F is SA2-type; that is, $F = A_1$ cop A_2 if A_1, $A_2 \in$ attr(R), then
  val($\sigma[F](R)$) := $\{t \mid t \in$ val(R) $\wedge\ t(A_1)$ cop $t(A_2)\}$
 (b) If F is (*not* F'), then
  val($\sigma[F](R)$:= val($R - \sigma[F'](R)$)
 (c) If F is ($F_1 \wedge F_2$), then
  val($\sigma[F](R)$) := val($\sigma[F_1](R) \cap \sigma[F_2](R)$)
 (d) If F is ($F1 \vee F2$), then
  val($\sigma[F](R)$) := val($\sigma[F_1](R) \cup \sigma[F_2](R)$) □

As in the 1NF algebra a selection with formula F selects a subset of tuples in val(R). A tuple t is in the result relation, iff F evaluates to true if attributes in F are substituted by the corresponding values of t. The only changes introduced are set comparisons and constant sets in connection with RV-attributes. We will extend the definition (a) of selection atoms in section 3.4.1 but keep the definitions (b), (c), (d).

As in the 1NF case we do not state context conditions explicitely here and in the following, e.g. for "$A1$ cop $A2$" in (2.a.ii). We assume that dom($d(A1)$) and dom($d(A2)$) are comparable w.r.t. cop and thus regard the selection as being undefined for noncomparable domains.

As a first example of a selection with set comparisons we could be interested in the department the administrational employee with the number 121 and job description "library" works in:

Q1 := σ[AE $\supseteq$ {⟨121, library⟩}] (DEPT)

The result Q1 contains the whole tuple of department 1.

3.3 *Schema transforming operations*

1NF algebra has two basic operators transforming relational schemata: Cartesian product and projection. NF^2 algebra introduces the two additional operators nest and unnest.

3.3.1 *Cartesian product.* The definition of the Cartesian product applies to NF^2 relations without change:

Definition 11 *"Cartesian product".* If $\text{sch}(R1) \cap \text{sch}(R2) = \varnothing$, then

(1) $\text{sch}(R1 \times R2) := \text{sch}(R1) \cup \text{sch}(R2)$

(2) $\text{val}(R1 \times R2) := \{t \mid t(\text{attr}(R1)) \in \text{val}(R1) \wedge t(\text{attr}(R2)) \in \text{val}(R2)\}$ □

Notice that we assume $\text{sch}(R1) \cap \text{sch}(R2) = \varnothing$ otherwise we must perform renaming (seee 3.3.2).

3.3.2 *Projection.* The definition of the projection for 1NF relations can be applied to NF^2 relations without change. But as attribute values in NF^2 relations may be relations in turn, this form of projection is not powerful enough for RV-attributes, since we can take an attribute value as a whole or not at all. In order to prepare for a major extension of the projection (section 3.4.2) we give a formal definition of the common projection in a modular, unusual way and we introduce a renaming facility.

Definition 12 *"projection list".* If we consider a projection $\pi[L](R)$ we call L the projection list, which has the following form:

$$L = A_{i1}:B_1, A_{i2}:B_2, \ldots, A_{in}:B_n$$

where the A_{ij} are attributes of R and the B_j are new names for the projected attributes. Again the ordering of the A_{ij} is arbitrarily fixed. For a given L as above we define

$$L^1 := A_{i1}, \ldots, A_{in}$$

$$L^2 := B_1, \ldots, B_n$$

that is, L^1 contains the projected attributes $A_{ij} \in \text{attr}(R)$ and L^2 contains new names for these attributes in the resulting relation. The ordering in L^1 and L^2 is the same as in L. □

Definition 13 *"projection".* We give a recursive definition which distinguishes two cases:

(1) The projection list has only one element, that is, $L = A_i:B$. If $A_i \in \text{attr}(R)$, then

(i) $\text{sch}(\pi[L](R) := \{(B, \text{sch}(d(A_i)))\}$

(ii) $\text{val}(\pi[L](R) := \{t \mid (\exists)\, r \in \text{val}(R) \wedge r(A_i) = t(B)\}$

(2) The projection list has more than one element, that is, $L = L', A_i:B$. If $A_i \in \text{attr}(R)$, then

(i) $\text{sch}(\pi[L](R)) := \text{sch}(\pi[L'](R)) \cup \text{sch}(\pi[A_i:B](R))$

(ii) $\text{val}(\pi[L](R)) := \{t \mid (\exists)\, r \in \text{val}(R) \wedge r(L'^1) \in \text{val}(\pi[L'](R)) \wedge r(A_i) \in \text{val}(\pi[A_i:B](R)) \wedge t(L'^2) = r(L'^1) \wedge t(B) = r(A_i)\}$ □

This definition seems too complicated. But we will later be able to replace simple elements in the projection list by whole relational expressions (section 3.4.2) by extending only part (1) of the above definition.

The new names for projected attributes are a mean of renaming. This will be necessary when the complex NF^2 projections are considered. If renaming is not necessary, we allow "A_{ij}" as an abbreviation for "$A_{ij}:A_{ij}$" in the projection list.

3.3.3 *Nest and unnest.* Nest (ν) and Unnest (μ)[9] change the height of the schema tree; especially they transform between 1NF and NF^2 relations. Assume a relation R with attributes $A_1, \ldots, A_n$. We define the nest operator ν applied to R along the last k attributes in the above ordering. This is no loss of generality because the ordering is arbitrary.

Definition 14 *"nest".* Let R be a relation and $\text{attr}(R) = \{A_1, \ldots, A_n\}$, $A' \in \textbf{NAME}$, $A' \neq A_i$, $i = 1, \ldots, n$. Let further $\textbf{A1} := A_1, \ldots, A_{n-k}$ and $\textbf{A2} := A_{n-k+1}, \ldots, A_n$.

Then $\nu[A_{n-k+1}, \ldots, A_n:A'](R)$ abbreviated by $\nu[\textbf{A2}:A'](R)$ is defined as follows:

(1) $\text{sch}(\nu[\textbf{A2}:\text{A}'](R)) := \text{sch}(R) - \{d(A_j) \mid j = n-k+1, \ldots, n\} \cup \{(A', \{\text{sch}(d(A_j)) \mid j = n-k+1, \ldots, n\})\}$

(2) $\text{val}(\nu[\textbf{A2}:A'](R)) := \{t \mid (\exists)\, r \in \text{val}(\pi[\textbf{A1}](R)) \wedge t(\textbf{A1}) = r(\textbf{A1}) \wedge t(A') = \text{val}(\pi[\textbf{A2}](\sigma[A_1 = r(A_1) \wedge \ldots \wedge A_{n-k} = r(A_{n-k})](R)))\}$ □

From this definition it can easily be seen, that the nest operator takes a set of attributes and forms a new RV-attribute out of these (see Fig. 4).

The schema transformation performed by nesting can be described by the schema tree: a new node is inserted as a direct descendent of the root. A set of formerly direct descendents of the root become descendents of this new node.

Next the unnest operator on R with respect to

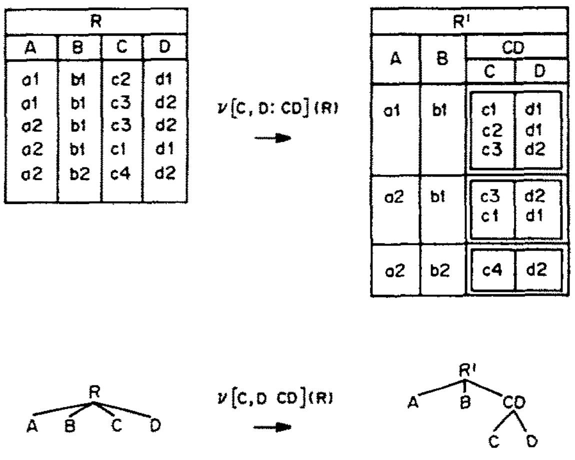

Fig. 4. Effect of nesting.

an attribute A_n is defined:

Definition 15 "*unnest*". Let R be a relation and $\text{attr}(R) = \{A_1, \ldots, A_n\}$, $A'_{n1}, \ldots, A'_{nk} \in \textbf{NAME}$, $\{A_1, \ldots, A_{n-1}\} \cap \{A'_{n1}, \ldots, A'_{nk}\} = \emptyset$.

If $\text{attr}(d(A_n)) = \{A_{n1}, \ldots, A_{nk}\}$, $k \geq 1$, then

(1) $\text{sch}(\mu[A_n : A'_{n1}, \ldots, A'_{nk}]\,(R))$
$:= \text{sch}(R) - \{d(A_n)\}$
$\cup \{\langle A'_{ni}, \text{sch}(d(A_{ni}))\rangle \mid 1 \leq i \leq k\}$

(2) $\text{val}(\mu[A_n : A'_{n1}, \ldots, A'_{nk}]\,(R))$
$:= \{t \mid (\exists)\, r \in \text{val}(R)$
$\wedge\, t(A_1, \ldots, A_{n-1}) = r(A_1, \ldots, A_{n-1})$
$\wedge\, t(A'_{n1}, \ldots, A'_{nk}) \in r(A_n)\}$

If $\text{attr}(d(A_n)) = \emptyset$, then

$$\mu[A_n : A'_n]\,(R) := \pi[A_1, \ldots, A_{n-1}, A_n : A'_n]\,(R)$$

□

The schema transformation of unnest can be described as eliminating a direct descendent of the root of the schema tree and connecting the direct descendents of the removed node directly to the root. If the node to be removed has no children, then no transformation is performed except renaming.

As shown in [9] the unnest operation is inverse to the nest operation. However, first unnesting can not generally be reversed by nesting (also see [12]).

3.4 *Nested application of algebraic operations*

The idea of the nested relational algebra is to apply some of the operators defined so far also to (RV-) attributes. As attributes occur in selection atoms and projection lists, selection and projection become the entry points for the nested algebra. The nested application of algebraic operators can easily be defined using recursive definitions for the value function. The termination of this recursion is guaranteed by the fact that each recursive call of the value function corresponds to descending one level in the schema tree. Because any schema tree has a finite depth the recursion ends when the leaf nodes are reached.

A side remark to the definitions we have chosen is in place here: we do *not* assume that every NF^2 relation can be obtained by a sequence of nest operations. Therefore, we cannot define the nested relational algebra by unnesting the involved relations into the 1NF form, applying some basic algebraic expressions and finally converting the result back to NF^2 relations. Furthermore, we would arrive at the problem of null values generated by unnesting empty sets. These problems are treated in [25]. For our definitions we decided to use the "direct NF^2 approach" presented below.

3.4.1 *Extended selection.* Any selection which does not specify the value of an attribute as a whole could not be formulated in selection atoms until now. In order to allow conditions for single tuples in an RV-attribute we apply selections and projections to attributes in selection atoms.

Obviously, also other operations could be useful within selection formulae. However, as will be seen later (section 3.5.2), the extended selection can be defined via simple selections and extended projection. Therefore we only give explicit definitions for $\sigma - \sigma$ and $\sigma - \pi$:

Definition 16 "*extended selection atoms*". The two types of selection atoms from definition 8 are extended by the following two classes which are only applicable to RV-attributes.

Type SA3: ($\sigma - \sigma$) "attribute" may be substituted by "$\sigma[F']$ (attribute)" in SA1 and SA2
Type SA4: ($\sigma - \pi$) "attribute" may be substituted by "$\pi[L]$ (attribute)" in SA1 and SA2 □

On the syntactical level algebraic operations are applied to attributes (that is, names) instead of relations within the nested formulations. Therefore we must define which relations have to be taken for the semantical interpretation.

We have to add two subcases to the definition of the value of a selection given in definition 10, (2a). According to the nested structure the value of a $\sigma - \sigma$- or $\sigma - \pi$-operation is defined via recursive calls of the value function val.

Definition 17 "*extended selection*". Definition 10 (1) and (2b–d) remain unchanged. Two subcases (iii) and (iv) are added to (2a):

(iii) F is SA3-type ($\sigma - \sigma$); that is, F contains a selection applied to an RV-attribute: at any place in (i) and (ii) substitute "$t(A)$" and "$t(Ai)$" by "$\text{val}(\sigma[F']\,(\langle d(A), t(A)\rangle))$" and "$\text{val}(\sigma[F']\,(\langle d(Ai), t(Ai)\rangle))$" respectively.

(iv) F is SA4-type ($\sigma - \pi$); that is, F contains a projection applied to an RV-attribute: at any place in (i) and (ii) substitute "$t(A)$" and "$t(Ai)$" by "$\text{val}(\pi[L]\,(\langle d(A), t(A)\rangle))$" and "$\text{val}(\pi[L]\,(\langle d(Ai), t(Ai)\rangle))$" respectively. □

In order to keep the definitions readable we did not give the explicit definitions for all of the possible operations combining SA1- and SA1-type with SA3- and SA4-type selections. One sample explicit definition, a $\sigma - \sigma$-operation for a SA1-type formula, is added:

(1) $\text{sch}(\sigma[\sigma[F']\,(A) \text{ cop } C]\,(R)) := \text{sch}(R)$
(2) $\text{val}(\sigma[\sigma[F']\,(A) \text{ cop } C]\,(R) := \{t \mid t \in \text{val}(R)$
$\wedge\, \text{val}(\sigma[F']\,(\langle d(A), t(A)\rangle)) \text{ cop } C\}$

Notice that operations are applied to attribute values in the selection formula to determine the selected tuples but nevertheless $\sigma - a$- and $\sigma - \pi$-operations select *whole* tuples out of $\text{val}(R)$!

As an example for a $\sigma - \sigma$-type operation consider Q2 asking for research departments employing translators:

$$\text{Q2} := \sigma[\text{DN} = \text{research} \wedge \sigma[\text{AJD} = \text{translation}]\,(\text{AE}) \neq \emptyset]\,(\text{DEPT})$$

The use of $\sigma - \pi$ is illustrated in the following query Q3 selecting departments having at least two technical employees one doing research and the other planning:

Q3 :=

$\sigma[\pi[\text{TJD}]\,(\text{TE}) \supseteq \{\text{research, planning}\}]\,(\text{DEPT})$

In either case, Q2 and Q3, the department 1 is the only result tuple.

It should be noted here that by means of $\sigma - \sigma$ and comparisons "$\neq\emptyset$" we can easily formulate existential quantifications. Also universal quantification can be formulated as shown by the following query $Q4$ asking for departments whose administrational staff consists of secretaries only:

$$Q4 := \sigma[\pi[\text{AJD}]\,(\text{AE}) = \{\text{secretary}\}]\,(\text{DEPT})$$

In our example, however, the result would be an empty relation with the schema sch(DEPT).

Using an extended notion of constants (cf. section 3.4.3) enables us to formulate more sophisticated conditions in the universal quantification, e.g. the relational division operator.

3.4.2 *Extended projection.* As the projection is a schema transforming operation involving specification of attributes we decided to introduce major extensions to this operator as a means of manipulating a complex NF^2 schema. It would be unsatisfactory if one could only take an $\bar{R}V$-attribute as a whole or nothing of it by projections. Parts of these subrelations should be selectable. We can distinguish two cases: not all tuples ($\pi - \sigma$) are selected or not all attributes ($\pi - \pi$) are specified. There are some more schema transformations that should be supported. We decided to let the projection (π or π- . . . -π) be the "navigator" in the NF^2 schema which allows the application of any unary operation to any node in the schema tree including nest ($\pi - \nu$) and unnest ($\pi - \mu$).

The next problem to be solved was the generalization of joins: with the extensions of the projection and of the selection described so far we are able to do transformation of single NF^2 relations. But how can two of them be combined? Cartesian product (and thus join) merely enables us to combine two relations at the outermost level. This is not sufficient; we should be able to join a relation also at a lower level in the schema of another. Several alternatives were discussed among them was the consideration to extend the definition of the Cartesian product. But, instead, we decided to allow binary operations within the projection list (that is, $\pi - \text{X}, \pi - \cup, \pi - -$). Since we can reach any node in the schema tree by a sequence of $\pi - \pi$ expressions we are able to combine the values of two attributes or an attribute and a constant (relation) by binary relational operators. This seems to be restrictive but the notion of a constant relation we use is very general (see next section).

Definition 18 *"extended projection lists".* Now the elements in a projection list L (see definition 12) of $\pi[L]\,(R)$ may have one of the following formats (A_i, A_k are attributes of R, $C \in \text{dom}(d(A_i))$ is a constant):

(1) (π – type) A_i:B (the only one up to now)
(2) ($\pi - \pi$-type) $\pi[L'](A_i)$:B
(3) ($\pi - \sigma$-type) $\sigma[F](A_i)$:B
(4) ($\pi - \nu$-type) $\nu[\ldots](A_i)$:B
(5) ($\pi - \mu$-type) $\mu[\ldots](A_i)$:B
(6) ($\pi - \cup$-type) $(A_i \cup A_k)$:B or $(A_i \cup C)$:B
(7) ($\pi - -$-type) $(A_i - A_k)$:B or $(A_i - C)$:B
(8) (π – X-type) $(A_i \times A_k)$:B or $(A_i \times C)$:B □

The definition (13) of the projection operator has to be extended accordingly. This is not difficult because we must only give new definitions for single-element projection lists due to our modular definition (cf. section 3.3.2).

Definition 20 *"extended projection".*

(1) $L = A_i$:B see definition 13

(2–5) to avoid unnecessary distinctions we use the following notation:
Let uop $\in \{\pi[L'], \sigma[F], \nu[\ldots], \mu[\ldots]\}$ and $L = \text{uop}(A_i)$:B, $\text{sch}(d(A_i)) \neq \emptyset$. Further let $v_i \in \text{dom}(d(A_i))$ be an arbitrary value. Then
 (i) $\text{sch}(\pi[L]\,(R)) := \{\langle B, \text{sch}(\text{uop}(\langle d(A_i), v_i\rangle))\rangle\}$
 (ii) $\text{val}(\pi[L]\,(R)) := \{t \mid (\exists)\, r \in \text{val}(R) \wedge t(B) = \text{val}(\text{uop}(\langle d(A_i), r(A_i)\rangle))\}$

(6–8) now let bop $\in \{\cup, -, \text{X}\}$, $v_i \in \text{dom}(d(A_i))$ and $v_k \in \text{dom}(d(A_k))$. We must distinguish two cases:
 (a) $L = (A_i \text{ bop } A_k)$:$B$, $\text{sch}(d(A_i))$, $\text{sch}(d(A_k)) \neq \emptyset$. Then
  (i) $\text{sch}(\pi[L]\,(R)) := \{\langle B, \text{sch}(\langle d(A_i), v_i\rangle \text{ bop } \langle d(A_k), v_k\rangle)\rangle\}$
  (ii) $\text{val}(\pi[L]\,(R)) := \{t \mid (\exists)\, r \in \text{val}(R) \wedge t(B) = \text{val}(\langle d(A_i), r(A_i)\rangle \text{ bop } \langle d(A_k), r(A_k)\rangle)\}$
 (b) $L = (A_i \text{ bop } C)$:B, $\text{sch}(d(A_i)) \neq \emptyset$.
  (i) $\text{sch}(\pi[L]\,(R)) := \{\langle B, \text{sch}(\langle d(A_i), v_i\rangle \text{ bop } C)\rangle\}$
  (ii) $\text{val}(\pi[L]\,(R)) := \{t \mid (\exists)\, r \in \text{val}(R) \wedge t(B) = \text{val}(\langle d(A_i), r(A_i)\rangle \text{ bop } C)\}$ □

Let us consider two examples of extended projections for our sample DEPT relation (Fig. 3):

We are interested in a relation Q5 comprising information about the technical staff of all departments but we do not want to see the employees' courses ($\pi - \pi$-type):

$$Q5 := \pi[D, \text{DN}, \pi[\text{TN}, \text{TJD}](\text{TE}):\text{TE}'](\text{DEPT})$$

Here we see the use of the renaming facility: as the schema of the subrelation TE is not the same as the one of TE′ in Q5 and because we identify the schema of an attribute by its name, we have to assign a new name to the result of the inner projection.

If we are interested in the maintenance personnel of each department, we can get this information by the following query Q6 (π – σ-type):

Q6 := $\pi[D$, DN,
σ[TJD = 'maintenance'] (TE):ME]
(DEPT)

Notice that the renaming of TE to ME was optional in this case.

As a comprehensive overview of the schema transformations caused by several operations consider the following illustration (Fig. 5).

3.4.3 *The notion of constants*. In the 1NF case, all elements of the basic domain D are regarded as allowed constants for the selection atoms of type SA1. These constants are specified in algebraic expressions explicitly. In our case of higher order domains we would be able to specify constant sets. For example "$\sigma[\pi[C\#]$ (COURSES) $\supseteq \{1, 2, 3\}]$ (EMP)" is a well-formed selection with the constant set "{1, 2, 3}" given explicitly by enumeration. Obviously, specifying set-valued constants explicitly can be very lengthy. Therefore, further solutions would be suitable for this problem. The advantage of constants, as far as query evaluation is concerned, is that tuples can be checked against predetermined values which are independent of the considered tuples. Hence, variables that have been set to some (constant) value could be used in algebraic expressions instead of explicitly specified constants. We want to go even one step further. Imagine a query on an NF^2 relation to be evaluated using an hierarchical (depth first) scan over the set of tuples. When we descend from the hierarchical level k to $k+1$ in this scan, we fix the values of the attributes on level k. Therefore, the attributes of level k can be regarded as "dynamic" constants (or variables with constant values) throughout the following part of the scan referring to the entered subrelation.

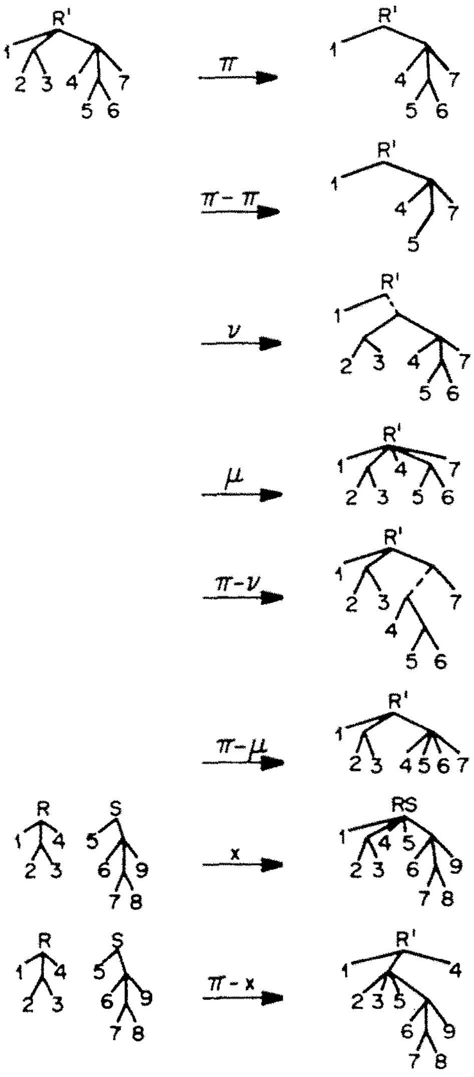

Fig. 5. Schema transforming operations of the NF^2 algebra.

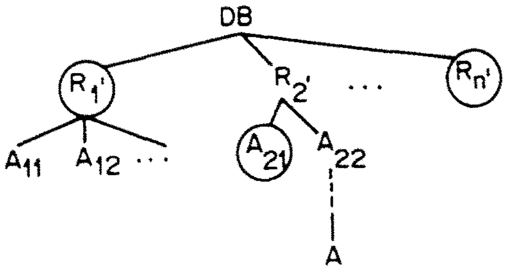

Fig. 6. Constants w.r.t. attribute A in the "universal" **DB** relation.

So we arrive at an extended notion of constants according to the following definition.

Definition 21 *"constants"*.

(a) "Static Constants":
Each element of a domain C out of the set **C** (cf. section 2.1) is a static constant

(b) "Dynamic Constants":
We assume a given set **DB** of relations R_1, . . . , R_n forming an "NF^2 relational database". We may represent this fact by a "universal" relation **DB** which has $R_1' = \text{nam}(R_1), \ldots, R_n' = \text{nam}(R_n)$ as attributes. **DB** contains a single ("universal") tuple **u** and we obtain the original relations from this new view by $R_i = \langle d(R_i'), u(R_i')\rangle$. (see Fig. 6)
Now let A be any node in the schema tree of **DB**. The set of dynamic constants DC with respect to node A in the tree **DB**, abbreviated by $DC(A)$ is the union of all attribute values of brother nodes of any predecessor of A.

Consequently, $DC(\mathbf{DB}) = \varnothing$ since there is no predecessor of **DB**; also $DC(R_i') = \varnothing$ since the predecessor of any R_i' is **DB** which has no brother. However, if A is an attribute of R_i', $DC(A)$ contains the values of the relations $R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n$. When we descend one level deeper and assume A_{ij} to be an attribute in $d(A_i)$ we obtain $DC(A_{ij})$ by $DC(A_i)$ and additionally the values of all brother nodes of A_i: that is, the nodes attr($d(A_i)$) − $\{A_i\}$. Constants w.r.t. A are circled in Fig. 6.

3.5 *Towards a minimal set of NF^2 operators*

In this section we first show how we can define the meaning of nested expressions. We will also show that the nested expressions in a selection formula can be replaced by nested expressions in a projection list.

3.5.1 *Expressions in nested algebraic operators.* With the given definitions of the basic operators we can recursively define the meaning of any expression; that is, any series of subsequently applied operators. Each algebraic operator results in an NF^2 relation and hence an expression can be defined via an induction over its finite (!) structure.

It seems to be useful to allow expressions not only on the outermost level but also within nested applications of the algebra. As an example, consider the natural join within a projection list. Undoubtedly this would be a convenient operation which could be defined as π-$\pi(\sigma(x))$. Up to now, expressions within the π-list are not permitted. Any kind of expression within a projection list can be defined via an expression at the outermost level containing no nested expressions; that is, "iteration" of the algebra can always be transformed to the outermost level. The way we do that is best explained by the following example:

Let E be an expression of the form E = π[expr(A_i):A] (R) and expr = op(expr′(A_i))

The we define E to be equivalent to

$$E = \pi[\text{op}(A'):A]\ (\pi[\text{expr}'(A_i):A']\ (R))$$

Notice, that expr′ is "shorter" than expr. By induction over the number of operators in E and inspection of all cases for op we conclude that expressions within a projection list do not add expressiveness. But they are a convenient abbreviation that can be defined formally via the above decomposition.

As an example for the use of a nested expression consider the following query Q7 asking for a relation showing technical and administrational employees for each department who share the same job description. (Q7 is a π-| X |-type query expressend here as π-$\pi(\sigma(X))$)

$$Q7 := \pi[D, \text{DN}, (\pi[\text{AN}, \text{TN}, \text{TJD}]\ (\sigma[\text{AJD} = \text{TJD}](\text{AE} \times \text{TE}))):\text{ATE}](\text{DEPT})$$

According to our previous discussion this expression is equivalent to

$$H1 := \pi[D, \text{DN}, (\text{AE} \times \text{TE}):\text{ATE}']\ (\text{DEPT})$$
$$H2 := \pi[D, \text{DN}, \sigma[\text{AJD} = \text{TJD}]\ (\text{ATE}')]\ (H1)$$
$$Q7 := \pi[D, \text{DN}, \pi[\text{AN}, \text{TN}, \text{TJD}]\ (\text{ATE}'):\text{ATE}]\ (H2)$$

Let us consider one additional query indicating how binary operations can be directed to relations deeper in the schema: Imagine a (1NF) relation CD($C\#$, CDES) containing descriptions of the courses attended by our technical personnel. We want to get this information into the DEPT relation. The query Q8 does that by joining the "outer" relation CD with any C relation in DEPT. Notice that CD is a constant w.r.t. node C.

$$Q8 := \pi[D, \text{DN}, \pi[\text{TN}, \text{TJD}, \pi[\text{CN}, \text{CDES}, Y]\ (\sigma[\text{CN} = C\#](C \times \text{CD})):\text{TCD}](\text{TE}):\text{TEC}](\text{DEPT})$$

As can be seen from this sample expression, we locate any node in a relation schema tree by a multiple nested $\pi - \pi - \pi$- . . . operation and apply the usual algebraic operators within the appropriate nesting level.

3.5.2 *Omitting extended selection atoms.* Using the construction rule from the previous section and an additional "trick", we can see that $\sigma - \sigma$ and $\sigma - \pi$ are not essential but can be defined via $\sigma - \sigma/\pi - \pi$ in connection with a simple selection and projection. The only problem is that $\sigma - \sigma$ and $\sigma - \pi$ select *whole* tuples, while $\pi - \sigma$ and $\pi - \pi$ change the subrelations. To overcome this difficulty we project all attributes of the relations and additionally the one which is modified by σ or π. Next, a simple selection refers to this new attribute. Finally, the additional attribute is eliminated by a simple projection. For instance:

$\sigma[\sigma[F']\ (A_i)$ cop $C]\ (R)$, with $A_i \in$ attr(R) and $C \in$ dom($d(A_i)$) is equivalent to $\pi[\text{attr}(R)]\ (\sigma[A_i'$ cop $C]\ (\pi[\text{attr}(R), \sigma[F']\ (A_i):A_i']\ (R)))$

Generally, we define a selection with a general expression expr in its formula by the following equation

$$\sigma[\text{expr cop } C]\ (R) := \pi[\text{attr}(R)]\ (\sigma[A' \text{ cop } C]\ (\pi[\text{attr}(R), \text{expr}:A']\ (R)))$$

This shows that the extension of selection atoms is unnecessary. The only essential extension consists in the nested application of the seven basic operations (see section 3.1) within the projection list.

4. CONCLUSIONS

We extended the usual notion of schema, value, attribute and domain of 1NF relations to relations with relation-valued attributes by applying them recursively. The usual definitions are contained as a special case. The relational algebra, first extended by two new operators nest and unnest, gets more effective by allowing algebraic expressions also for the relations within relations. In order to find simple formal definitions for such nested expressions, we first defined primitives (section 3.2, 3.3) in such a way that we could apply recursive definitions (section 3.4). As shown in section 3.5, the essential extension we introduced was the new projection operator. Expressions in selection formulae can be defined via expressions in projection lists. Special attention was given to the definition of constants (section 3.4.3).

We have not yet solved the problem of equivalent expressions and the question of algebraic optimization on nested expressions. Both areas must be considered when we follow the plan of using this data model for a description of internal data structures and for a description of related mappings. The formalism of this paper gives the solid basis for further research on this subject. As we see this relational algebra at an internal level we did not consider the aspects of a user query language nor did we touch the aspects of integrity constraints and design theory. A protype database system based on our model is currently being implemented at the Technical University of Darmstadt.

Acknowledgement—We are grateful for many discussions with G. Jaeschke and P. Pistor which have influenced further investigations since the reports[9, 5] have been written. While G. Jaeschke tries to define recursive relational algebra expressions in a VDM-like approach[26] we have decided to define the meaning of general nested expressions bottom-up by sequences of a few basic building blocks. We also thank the referees for their constructive comments.

REFERENCES

[1] E. F. Codd. A Relational Model for Large Shared Data Banks, *Comm. ACM* **13**(6) (1970).

[2] C. J. Date. *An Introduction to Database Systems* (3rd ed.) Addison-Wesley, Reading, Mass. (1981).

[3] P. P. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM TODS* 1(1) (1976).

[4] D. Maier. *The Theory of Relational Databases* Computer Science Press, Rockville, Maryland (1983).

[5] H.-J. Schek and P. Pistor. Data Structures for an Integrated Database Management and Information Retrieval System. *Proc. VLDB Conf.* Mexico City, Mexico (1982).

[6] H.-J. Schek and M. H. Scholl. The NF^2 Relational Algebra for a Uniform Manipulation of the External, Conceptual and Internal Data Models (in German). in *Sprachen fuer Datenbanken* (J. W. Schmidt ed.), Informatik Fachberichte Nr. 72, Springer, Berlin (1983).

[7] Y. Kambayashi, K. Tanaka and K. Takeda. Synthesis of Unnormalized Relations Incorporating More Meaning. *Int. Journal of Information Sciences* (Special Issue on Databases) (1983).

[8] H. Arisawa, K. Moriya and T. Miura. Operations and Properties on Non-First-Normal-Form Relational Databases. *Proc. VLDB Conf.* Florence, Italy (1983).

[9] G. Jaeschke and H.-J. Schek. Remarks on the Algebra of Non-First-Normal-Form Relations. *Proc. 1st. ACM SIGACT/SIGMOD Symp. on Princ. of Database Systems* Los Angeles, Ca. (1981).

[10] J. Kobayashi. *An Overview of Database Management Technology* Techn. Report TRCS-4. SANNO College, Kanagawa, Japan (1981).

[11] A. Makinouchi. A Consideration on Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model. *Proc. VLDB Conf.*, Tokyo, Japan (1977).

[12] P. C. Fischer and S. J. Thomas. Operators for Non-First-Normal-Form Relations. *Proc. IEEE COMPSAC* (1983).

[13] S. Abiteboul and N. Bidoit. Non First Normal Form Relations to Represent Hierarchically Organised Data. *Proc. 2nd ACM SIGACT/SIGMOD Symp. on Princ. of Database Systems* Waterloo, Ontario, Canada (1984).

[14] T. Niemi. A Seven-Tuple Representation for Hierarchical Data Structures. *Inform. Systems* 8(3) (1983).

[15] S. Gibbs and D. Tsichritzis. A Data Modelling Approach for Office Information Systems. *ACM Trans. on Office Information Systems* **1**(4) (1983).

[16] N. C. Shu, V. Y. Lum, F. C. Tung and C. L. Chang. Specification of Forms Processing and Business Procedures for Office Automation. *IEEE Trans. on Softw. Eng.* **SE-8**(5) (1982).

[17] D. Luo and B. Yao. Form Operation by Example, A Language for Office Information Processing. *Proc. ACM SIGMOD Conf.* Ann Arbor, Mich. (1981).

[18] S. B. Navathe. Schema Analysis for Database Restructuring. *ACM TODS*. 5(2) (1980).

[19] D. Tsichritzis. Form Management. In *"Omega Alpha"*, (D. Tsichritzis ed.), Techn. Report CSRG-127, Univ. of Toronto (1981).

[20] R. Haskin and R. Lorie. On Extending the Functions of a Relational Database System. *Proc. ACM SIGMOD Conf.* Orlando, Fl. (1982).

[21] A. L. Furtado and L. Kerschberg. An Algebra of Quotient Relations. *Proc. ACM SIGMOD Conf.* Toronto, Canada (1977).

[22] W. Lamersdorf and J. W. Schmidt. Recursive Data Models (in German). in: *Sprachen fuer Datenbanken*, (J. W. Schmidt ed.), Informatik-Fachberichte Nr. 72, Springer, Berlin, (1983).

[23] F. Bancilhon, P. Richard and M. Scholl. On Line Processing of Compacted Relations. *Proc. VLDB Conf.* Mexico City, Mexico (1982).

[24] ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report 75-02-08. In *FDT-Bulletin of ACM SIGMOD* 7(2) (1975).

[25] P. Hoeller. *Null Values and their Integration into the NF^2 Relational Model* (in German), Diploma Thesis, Techn. Univ. of Darmstadt, (1985).

[26] G. Jaeschke. *Recursive Algebra for Relations with Relation-Valued Attributes* Technical Report 84.01.003, IBM Heidelberg Scientific Centre (1984).