INSTITUT FÜR PROGRAMMIERSPRACHEN UND INFORMATIONSSYSTEME
Technische Universität Carolo-Wilhelmina zu Braunschweig

Abteilung Datenbanken
Prof. Dr. H.-D. Ehrich

D-3300 Braunschweig, den 09.01.89

Gaußstraße 12
Postfach 3329
Fernruf (0531) 391-3271
Telex 952526

Informatik · TU Braunschweig · Postfach 3329 · D-3300 Braunschweig

University of Waterloo
Faculty of Mathmatics
Department of Computer Science
-Technical reports- Donna Mc. Cracken

Waterloo, Ontario
CANADA

Ladies and Gentlemen,

we would like to get one of your published reports on exchange basis:

Deshpande, V.; Larson, P.-A.: An algebraic for nested relations. Waterloo, 1987, Department of Computer Science, RR 87/65.

Thanks in advance.
B.o.

(J. Bleiß)

sent Jan. 24/89

Printing Requisition/Graphic Services

89040

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION: An Algebra for Nested Relations CS-87-65

DATE REQUISITIONED: Dec. 1/87 — DATE REQUIRED: ASAP — ACCOUNT NO. 1 2 6 6 5 4 3 4 1

REQUISITIONER PRINT: P.-A. [illegible] — PHONE: [illegible] — SIGNING AUTHORITY: [signature]

MAILING INFO — NAME: [illegible] — DEPT.: Computer Science — BLDG. & ROOM NO.: MC 6081E — [X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 28 — NUMBER OF COPIES: 100

TYPE OF PAPER STOCK: Alpac Ivory — 140M — 10x14 Glosscoat (stock)

PAPER SIZE: 10 Bu.

PAPER COLOUR: White — INK: Black — Holland Tint

PRINTING — NUMBERING FROM / TO

BINDING/FINISHING — COLLATING / STAPLING / HOLE PUNCHED / PLASTIC RING

FOLDING/PADDING: 7x10 saddle stitched — CUTTING SIZE

Special Instructions: Special cover cover with Black Ink Format.

COPY CENTRE — OPER. NO. / BLDG. / MACH. NO.

DESIGN & PASTE-UP — OPER. NO. / TIME / LABOUR CODE: D01, D01, D01

TYPESETTING — QUANTITY: PAP000000 T01; PAP000000 T01; PAP000000 T01

PROOF: PRF; PRF

Item	Quantity	Oper. No.	Time	Labour Code
NEGATIVES				
FLM [illegible]000		5	10	C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01
PMT				
PMT				C01
PMT				C01
PMT				C01
PLATES				
PLT [illegible]	1	5	80	P01
PLT 11700	14	1	28	P01
PLT				P01
STOCK				
KR[illegible] 1400	120	16	20	001
[illegible] 1400	910	1	105	001
				001
				001
BINDERY				
RNG				B01
RNG		24	5	B01
RNG		1	5	B01
MIS000000		[illegible]	[illegible]	B01

OUTSIDE SERVICES

$ COST

UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

An Algebra for Nested Relations

V. Deshpande
P.-Å. Larson

Research Report CS-87-65

December, 1987

An Algebra for Nested Relations

*V. Deshpande and P.-Å. Larson**
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1
Report CS-87-65

Abstract

In a nested relation attributes are allowed to have relations as values. In this paper we present an algebra for nested relations. The operators of traditional relational algebra are extended to nested relations and operators for nesting and unnesting are defined. To be able to combine and modify subrelations within a relation, a new operator called a subrelation constructor is introduced. The proposed algebra simplifies and generalizes earlier approaches to defining an algebra for nested relations.

1 Introduction

A fundamental assumption of traditional relational database theory and current relational database systems is that all relations are in First Normal Form (1NF), that is, all attributes must have atomic (non-decomposable) domains. Such relations are commonly called *flat relations*. There are, however, both theoretical and practical reasons for investigating databases where this assumption has been relaxed and attributes are allowed to have relations as values. A relation-valued attribute may, in turn, have attributes which have relations as values, and so on. We will call relations of this type *nested relations*, instead of the rather clumsy non-first-normal-form relations.

Attempts to extend the use of relational databases to non-traditional applications dealing with complex objects, for example, engineering designs or office forms, have had limited success. Even in more conventional business applications, the limitation to flat relations has proved to be a severe

*This research was supported by Cognos, Inc., Ottawa and by the Natural Sciences and Engineering Research Council of Canada under grant No. A-2460.
Authors' e-mail addresses: {vdeshpande, palarson}@waterloo.{edu,cdn,csnet}

EMPLOYEE

ENO	NAME	CHILDREN			TRAINING	
		NAME	DOB	SEX	CNO	DATE
105	John	Jane	800510	F	314	791010
		Eric	821005	M	606	810505
					714	820620
123	Anne	Maria	751112	F	315	810613
					423	820711
153	Bruce				314	791010
205	Ian	Bob	701016	M		
		Steve	750115	M		

Table 1: An employee database consisting of a single nested relation.

EMP

ENO	NAME
105	John
123	Anne
153	Bruce
205	Ian

CHILDREN

ENO	CNAME	DOB	SEX
105	Jane	800510	F
105	Eric	821005	M
123	Maria	751112	F
205	Bob	701016	M
205	Steve	750115	M

TRAINING

ENO	CNO	DATE
105	314	791010
105	606	810505
105	714	820620
123	315	810613
123	423	820711
153	314	791010

Table 2: The employee database structured as three flat relations.

restriction. It leads to unnecessary fragmentation of the database, both conceptually and physically. As a result, querying the database becomes cumbersome and slow because, except for the simplest queries, the required data has to be collected from several relations and joined to obtain the result. There are many situations where the insistence on flat relations leads to an artificial separation of data. The following example illustrates one such situation.

Consider a simple database containing information about employees, their children, and courses taken by the employees. If nested relations are allowed, this information can be stored in a single relation. An instance of this relation is shown in Table 1. (DATE refers to the date when the employee completed the course.) If nested relations are not allowed, the information must be separated into three flat relations: EMP, CHILDREN, TRAINING. This is shown in Table 2. For simplicity, we assume that the conceptual relations shown in Tables 1 and 2 are also physically stored in the same way.

There is little to be gained by structuring the database as three separate relations. At the conceptual level, we fail to see how the version in Table 2

would be easier to understand. There is no redundancy in the nested relation in Table 1 and no update anomalies can occur. Physically, the three-table version requires more storage because the attribute ENO is repeated in both CHILDREN and TRAINING.

Querying the database in Table 2 is also conceptually more difficult and actual processing of a query is often slower. This is typically caused by the need to join several relations. A join must be explicitly carried out in the flat relations case, whereas it is already materialized in the nested relation. Consider the following query:

Example 1.1 *Find* ENO *and* NAME *of all employees without children.*

In traditional relational algebra this would be expressed as

$$\begin{aligned} \text{T} &:= \pi[\text{ENO}](\text{EMP}) - \pi[\text{ENO}](\text{CHILDREN}) \\ \text{Result} &:= \text{T} \bowtie \pi[\text{ENO,NAME}](\text{EMP}) \end{aligned}$$

The translation of the query into relational algebra can hardly be called obvious. In the relational algebra for nested relations developed in this paper, the same query translates into

$$\text{Result} := \pi[\text{ENO,NAME}]\sigma[\text{CHILDREN} = \emptyset](\text{EMPLOYEE})$$

(The condition CHILDREN $= \emptyset$ simply states that an EMPLOYEE tuple qualifies if the (sub)relation CHILDREN is empty.) Evaluation of the first query is also more complicated and likely to be significantly slower than evaluation of the second query. The first query involves computing a set difference and a join, which are both slow and expensive operations. The second query can be evaluated in a single scan of the EMPLOYEE relation.

In this paper we define an algebra for nested relations. We build on earlier work by [RKS84], [SS86], [AB84a], and [AB84b]. Roth, Korth and Silberschatz [RKS84] did not define any operations on subrelations. These are handled in our algebra by a new operator called a *subrelation constructor.* Our algebra is closest to the one proposed by Schek and Scholl in [SS86]. We have tried to simplify the definitions of the operators by making them consistently recursive. Our algebra, we feel, is also easier to understand. The algebra proposed by Abiteboul and Bidoit [AB84a] for Verso-relations differs significantly from ours. Furtado and Kerschberg [FK77] proposed an algebra for 'quotient relations' which can be viewed as simple nested relations. Fischer and Thomas [TF86] studied the interaction of nest and unnest with other relational operators.

Many other aspects of nested relations have been studied. Extensions of SQL to nested relations have been proposed in [RKB85]. Normal forms for nested relations were first investigated by [M77] and later by several other researchers [RK87], [OY85].

The rest of the paper is organized as follows. In section 2 we introduce and define a number of basic concepts. In section 3 we define the effect of the traditional operators (union, intersection, difference, select, project, Cartesian product, join) when applied to nested relations. The operators nest and unnest are also defined, as is the rename operator. In section 4 we introduce the new concept of a subrelation constructor, and show several examples of queries posed using the constructor to illustrate the ease of use and power of the algebra. In section 5 we show the same queries posed in the algebra proposed by [SS86] as a way of comparison. Finally, section 6 contains a brief conclusion and prospects for further work.

2 Basic Concepts

In a traditional relational database, all attributes of a relation are atomic-valued. In a nested relation, we allow attributes to be *relation-valued.* This can continue recursively a finite number of times, that is, a relation-valued attribute may, in turn, contain relation-valued attributes. A relation-valued attribute is also called a *subrelation.* An attribute which is atomic-valued is called a *single-valued* attribute. We give the following recursive definition.

Definition 2.1 (Nested relation) *The scheme of a* nested relation *is of the form* $R(A_1, A_2, \ldots, A_n)$, $n \geq 1$, *where* R *is the* name *of the scheme and each* A_i *is either the name of a single-valued attribute (defined over some domain) or the scheme of a nested (sub)relation.*

A *flat relation* is a relation where all attributes are single-valued. A database scheme is a collection of relation schemes $R_1, R_2, \ldots, R_m$. An *instance* r of a relation R consists of *tuples* over the scheme R.

The scheme of a nested relation R can be represented by a tree, T_R, called the *scheme diagram.* This is a convenient way of graphically illustrating the nesting structure of the relation. T_R is a tree with R as the root node and the relation-valued attributes of R as subtrees. The nodes of the scheme diagram are labelled with the names of the corresponding (sub)relations. As a convention, all the attributes of the (sub)relation are listed to the right of a node. The scheme diagram of relation EMPLOYEE is shown in Figure 1.

Let R be the name of a relation or subrelation. Then attr(R) denotes the set of (single-valued and relation-valued) attribute names of R, sattr(R) the set of single-valued attribute names of R, and rattr(R) the set of relation-valued attribute names of R. For example, attr(EMPLOYEE) = {ENO, NAME, CHILDREN, TRAINING}, sattr(EMPLOYEE) = {ENO, NAME}, rattr(EMPLOYEE) = {CHILDREN, TRAINING}. In this paper, we also adopt the convention that Y denotes names of subrelations, X denotes single-valued attribute names, V, W and Z denote attribute names that may be

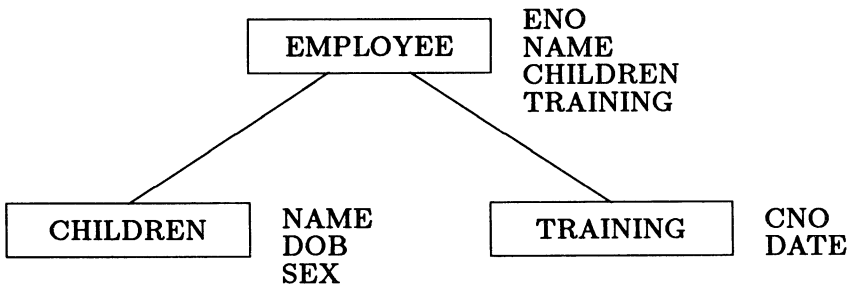


Figure 1: Scheme diagram of the relation EMPLOYEE

either single-valued or relation-valued. If $Z \subseteq$ attr(R) and t is a tuple over R, we write $t[Z]$ to denote the projection of t onto the attributes in Z.

The definition of a relational algebra operator must specify both the structure of the resulting relation and its value. To define the structure of a relation we introduce the concept of the *format* of a relation. Note that a relation created by an algebra expression does not have a name, unless it is explicitly assigned one. For example, the relation resulting from $\pi[X](R)$ does not have a name. Hence, the format of a relation does not include the top-level name of the relation, if it has one. However, if we want to be able to use the result of an algebra expression as an operand in another expression, we must be able to refer to its attributes, for example, in a selection condition. Hence, the format of a relation includes attribute names. The result relation inherits the attribute names of its operand relations, according to certain rules. A formal definition follows.

Definition 2.2 (Format) *Let X be a single-valued attribute, and $R(Z_1, \ldots, Z_n)$ a nested relation scheme where R is the name of the relation and $Z_1, \ldots, Z_n$ are attribute names.*

$$\begin{aligned} \text{format}(X) &= \text{':'domain}(X) \\ \text{format}(R(Z_1, \ldots, Z_n)) &= \text{'('}Z_1\text{format}(Z_1), \ldots, Z_n\text{format}(Z_n)\text{')'} \end{aligned}$$

Example 2.1

format(R(A,B(C,E)))
= (A:format(A), B format(B))
= (A:domain(A),B(C:format(C),E:format(E)))
= (A:domain(A),B(C:domain(C),E:domain(E)))
format(π[ENO,TRAINING](EMPLOYEE))
= (ENO:domain(ENO),TRAINING(
CNO:domain(CNO),DATE:domain(DATE)))

Definition 2.3 (Equivalence of Formats) *Let R_1, R_2 be two nested relations. The formats of R_1 and R_2 are equivalent, written* format(R_1) = format(R_2), *if*

$$\text{attr}(R_1) = \text{attr}(R_2) \wedge \forall A_i \in \text{attr}(R_1), B_j \in \text{attr}(R_2),$$
$$(A_i = B_j \Rightarrow \text{format}(A_i) = \text{format}(B_j))$$

Note that according to the above definition, the ordering of attributes within a relation is unimportant in determining the equivalence of relation structures. Attributes are identified by their names, not their positions.

We restrict the class of nested relations to those which are in *Partitioned Normal Form (PNF)*. A relation R is in PNF if the single-valued attributes of R form a key for the relation, and recursively, each relation-valued attribute of R is also in Partitioned Normal Form. For example, the instance of EMPLOYEE in Table 1 is in PNF. A formal definition is given below, slightly modified from [RKS84].

Definition 2.4 (Partitioned Normal Form) *Let R be a relation with attr(R) consisting of single-valued attributes X and relation-valued attributes Y. An instance r of R is in* PNF *if*

1. *$X \rightarrow XY$, (X functionally determines XY) and*
2. *For all $t \in r$ and for all $Y_i \in Y$, $t[Y_i]$ is in* PNF.

In the relational algebra to be defined, *pathnames* are used for referring to subrelations which are nested in the relation structure. We give the following definition.

Definition 2.5 (Pathname) *Let R be a nested relation. An expression of the form $Y_1.Y_2.\cdots.Y_k$ is a valid pathname in R if $Y_1 \in$* rattr(R) *and $Y_i \in$* rattr(Y_{i-1}), $i = 2, 3, \ldots, k$.

A pathname uniquely identifies a subrelation within R. Note that the relation name R is not included in the pathname. The *complete name* of an attribute Z in a subrelation Y_j of a relation R, is then of the form $Y_1.Y_2.\cdots.Y_j.Z$ where $Y_1.Y_2.\cdots.Y_j$ is the pathname identifying Y_j. By using complete names we can always uniquely identify an attribute within a relation. For example, in EMPLOYEE, CHILDREN.NAME refers to the attribute NAME of the subrelation CHILDREN, while NAME refers to the attribute NAME in EMPLOYEE.

Next we define the *scope* of a pathname. The scope consists of the set of attributes (names) which can be referred to when the pathname is specified. Intuitively, the scope consists of all the attributes which are 'seen' as one goes along the path starting at the root of the scheme diagram of R and going down to the subrelation identified by the pathname.

Definition 2.6 (Scope of a Pathname) *Let $S = \{R_1, R_2, \ldots, R_m\}$ be a database scheme and $P = Y_1.Y_2.\cdots.Y_k$ a valid pathname in relation R_j. Then the scope of P is defined as*

$$\begin{aligned}\text{scope}(P) \;=\; & S \cup \text{attr}(R_j) \cup \text{attr}(Y_1) \cup \text{attr}(Y_1.Y_2) \cup \ldots \cup \text{attr}(Y_1.Y_2.\cdots.Y_k) \\ & - \{R_j, Y_1, Y_1.Y_2, \ldots, Y_1.Y_2.\cdots Y_k\}\end{aligned}$$

The immediate scope of P refers to the set of attributes that are seen at the lowest level of the path, that is, iscope(P) = attr($Y_1.\cdots.Y_k$).

For example, consider the scheme R(A,B,C(D,E),H(I,J(K,L))) , and the path P=H.J . Then

$$\begin{aligned}\text{scope(H.J)} &= \{\text{R}\} \cup \text{attr(R)} \cup \text{attr(H)} \cup \text{attr(H.J)} - \{\text{R,H,H.J}\} \\ &= \{\text{R}\} \cup \{\text{A,B,C,H}\} \cup \{\text{H.I,H.J}\} \cup \{\text{H.J.K,H.J.L}\} - \{\text{R,H,H.J}\} \\ &= \{\text{A,B,C,H.I,H.J.K,H.J.L}\} \\ \text{iscope(H.J)} &= \text{attr(H.J)} \\ &= \{\text{H.J.K,H.J.L}\}\end{aligned}$$

At the instance level, each occurrence of J consists of a set of tuples (possibly empty) and everything above it along the path will have fixed values which can be referred to as needed. Note that it is not possible to (directly) refer to attributes inside C, since attributes D, E are not within the scope of P.

A running example of a University database is used in this paper to illustrate the proposed algebra. The database contains information about faculties, students, courses, and course offerings. An instance of the database is shown in Tables 3 - 6.

3 The Algebra Operators

In this section we extend the operators of traditional relational algebra to nested relations and define three additional operators: nest, unnest and rename. The operators nest and unnest were originally introduced in [JS82]. The definitions are similar to those in [RKS84]. Note, however, that we allow empty subrelations.

3.1 The Set Operators

The set operators union, intersection and difference are extended to allow set operations to take place between nested relations. The formats of the two operands taking part must be equivalent, and the resultant relation also has the same format as the operands.

FNAME	DEPARTMENT					
	DNAME	COURSES	PROFESSORS			
		CNO	PNO	PNAME	OFFICE	EXT
Mathematics	Math	MAT 130a	82035	J. Barnes	MC 5034	2698
		MAT 130b				
	Comp.Sci.	CS 240	80148	P. Smith	MC 6212	6678
		CS 340	79250	A. Curtis	MC 6342	6717
		CS 448	84079	R. Alegre	MC 6017	5712
		CS 488				
	CO	CO 230	83123	R. Sims	MC 5054	2765
		CO 330	81341	B. Marshall	MC 5115	7561
			75017	M. Weber	MC 5165	6304
Science	Physics	PHY 263	77015	R. Robin	PHY 363	4241
		PHY 354	82169	C. Theil	PHY 345	5112

Table 3: The relation FACULTY.

SNO	NAME	FAC
8380	D. Walters	Science
8470	A. Brown	Mathematics
8619	T. Lonsdale	Mathematics
8621	R. Helbig	Mathematics
8653	G. Wright	Mathematics

Table 4: The relation STUDENT.

CNO	CNAME	PREREQ
		CNO
MAT 130a	Calculus I	
MAT 130b	Calculus II	MAT 130a
CS 240	Prog. Principles	
CS 340	Data Structures	CS 240
		CO 230
CS 448	Database Mgmt.	CS 340
CS 488	Graphics	CS 340
		CO 330
		MAT 130b
CO 230	Combinatorics	
CO 330	Enumeration	CO 230
PHY 263	Classical Mech.	MAT 130a
		MAT 130b
PHY 354	Optics	MAT 130b

Table 5: The relation COURSES.

CNO	TERM	ENROLLMENT	
		SNO	GRADE
CS 240	F86	8653	85
		8621	80
CS 340	W87	8621	78
		8653	82
		8619	90
CS 448	S87	8621	83
		8653	88
		8619	92
CS 488	W88		
CO 230	W86	8380	67
		8470	93
CO 230	F86	8653	87
		8619	82
CO 330	W87	8621	47
		8619	92
		8653	80

Table 6: The relation OFFERINGS.

Let $\Theta \in \{\cup, \cap, -\}$, and R_1, R_2 relational expressions where format(R_1) = format(R_2). The syntax of the operators is:

$$R_1 \Theta R_2$$

We now look at the instance level effects of the operators. Let rattr(R_1) = rattr(R_2) = $(W_1, W_2, \ldots, W_n)$.

- Union:

$$\begin{aligned} r_1 \cup r_2 = \{s \mid (\exists t_1 \in r_1, \exists t_2 \in r_2 : s[\text{sattr}(R_1)] = t_1[\text{sattr}(R_1)] = t_2[\text{sattr}(R_2)] \\ \wedge s[W_1] = t_1[W_1] \cup t_2[W_1] \wedge \ldots \wedge s[W_n] = t_1[W_n] \cup t_2[W_n]) \\ \vee (\exists t_1 \in r_1,\ \forall t_2 \in r_2 : s = t_1 \wedge\ t_1[\text{sattr}(R_1)] \neq t_2[\text{sattr}(R_2)]) \\ \vee (\exists t_2 \in r_2,\ \forall t_1 \in r_1 : s = t_2 \wedge\ t_2[\text{sattr}(R_2)] \neq t_1[\text{sattr}(R_1)])\} \\ \text{format}(R_1 \cup R_2) = \text{format}(R_1) \end{aligned}$$

- Intersection:

$$\begin{aligned} r_1 \cap r_2 = \{s \mid \exists t_1 \in r_1, \exists t_2 \in r_2 : s[\text{sattr}(R_1)] = t_1[\text{sattr}(R_1)] = t_2[\text{sattr}(R_1)] \\ \wedge s[W_1] = t_1[W_1] \cap t_2[W_1] \wedge \ldots \wedge s[W_n] = t_1[W_n] \cap t_2[W_n]\} \\ \text{format}(R_1 \cap R_2) = \text{format}(R_1) \end{aligned}$$

- Difference:

$$\begin{aligned} r_1 - r_2 = \{s \mid \exists t_1 \in r_1 : (\forall t_2 \in r_2 : s = t_1 \wedge t_1[\text{sattr}(R_1)] \neq t_2[\text{sattr}(R_1)]) \\ \vee (\exists u_2 \in r_2 : s[\text{sattr}(R_1)] = t_1[\text{sattr}(R_1)] = u_2[\text{sattr}(R_1)] \wedge t_1 \neq u_2 \\ \wedge s[W_1] = t_1[W_1] - t_2[W_1] \wedge \ldots \wedge s[W_n] = t_1[W_n] - t_2[W_n])\} \\ \text{format}(R_1 - R_2) = \text{format}(R_1) \end{aligned}$$

The definitions are recursive, since the operators are applied to each subrelation $W_i \in$ rattr(R_1). Table 8 shows the results of applying the operators to the nested relations of Table 7.

As defined above, the set operators always return a result which is in PNF. Applying one of these operators directly does not always give the same result as first unnesting the operand relations, applying the operator, and renesting the resulting (flat) relation. When applying an operator directly, the result may contain tuples with empty subrelations. Nesting a flat relation cannot produce empty subrelations (unless null values are allowed). If the

R_1

X_1	Y_1		
	X_2	Y_2	
		X_3	X_4
1	1	1	1
		1	2
	2	3	1
	3	1	2
		2	2
		3	2
2	2	1	2
		2	1
	3	1	1
3	1	3	1
		2	2
		2	1
4			

R_2

X_1	Y_1		
	X_2	Y_2	
		X_3	X_4
1	2	3	1
		2	2
		1	1
	3	1	2
		2	2
		3	3
3	1	2	1
		1	3
		1	1
	2	1	3
4	1	1	2
		2	3

Table 7: The given instances R_1 and R_2.

result does not contain any tuples with empty subrelations, then it is the same as would be obtained by first unnesting for union and intersection, but not for difference. Unnesting first and then applying set difference may produce a result which is not in PNF.

3.2 Projection

The project operator is extended to allow projection on subrelations as well as single-valued attributes. The syntax is

$$\pi[Z](R)$$

where R is an arbitrary relational expression, and Z is a set of attributes such that $Z \subseteq \text{attr}(R)$.

We now look at the instance level effect of the operator. Let $Z = \{Z_1, \ldots, Z_k\} \subseteq \text{attr(R)}$. We give two different interpretations of the operator: with and without preservation of PNF.

Projection without preservation of PNF: There are two cases to consider.

$R_1 \cup R_2$

X_1	Y_1		
	X_2	Y_2	
		X_3	X_4
1	1	1	1
		1	2
	2	3	1
		2	2
		1	1
	3	1	2
		2	2
		3	2
		3	3
2	2	1	2
		2	1
	3	1	1
3	1	3	1
		2	2
		2	1
		1	3
		1	1
	2	1	3
4	1	1	2
		2	3

$R_1 \cap R_2$

X_1	Y_1		
	X_2	Y_2	
		X_3	X_4
1	2	3	1
	3	1	2
		2	2
3	1	2	1
4			

$R_1 - R_2$

X_1	Y_1		
	X_2	Y_2	
		X_3	X_4
1	1	1	1
		1	2
	2		
	3	3	2
2	2	1	2
		2	1
	3	1	1
3	1	3	1
		2	2
4			

Table 8: Union, intersection and difference applied to R_1, R_2.

R

X_1	X_2	Y_1 A	Y_1 B
1	1	1	2
		2	2
1	2	1	1
		1	2
2	1	2	1
2	2	1	1
		2	1

(a)

$\pi[X_1, Y_1](R)$

X_1	Y_1 A	Y_1 B
1	1	2
	2	2
1	1	1
	1	2
2	2	1
2	1	1
	2	1

(b)

$\pi[X_1, Y_1](R)$

X_1	Y_1 A	Y_1 B
1	1	2
	2	2
	1	1
2	2	1
	1	1

(c)

Table 9: Projection and preservation of PNF for case 1.

1. sattr(Z) $\neq \emptyset$, that is, Z contains at least one single-valued attribute.

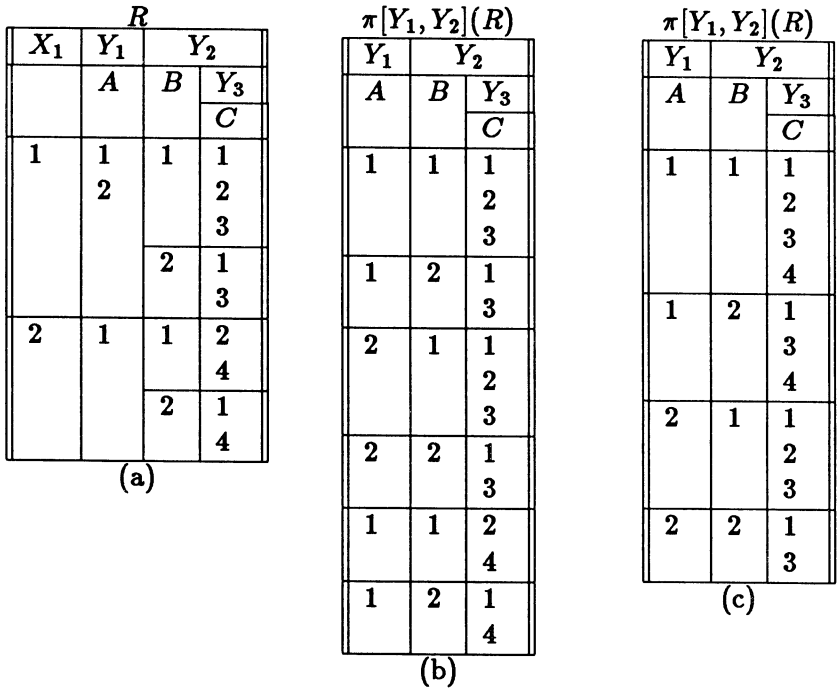
$$\begin{aligned} \pi[Z](r) &= \{s \mid \exists t \in r : s = t[Z]\} \\ \text{format}(\pi[Z](R)) &= (Z_1\text{format}(Z_1), \ldots, Z_k\text{format}(Z_k)) \end{aligned}$$

2. sattr(Z) $= \emptyset$, that is, Z contains only relation-valued attributes. Assume that $attr(Z_i) = \{Z_{i_1}, \ldots, Z_{i_l}\}$, $1 \leq i \leq k$. It is required that $\text{attr}(Z_i) \cap \text{attr}(Z_j) = \emptyset$ for $i \neq j$.

$$\begin{aligned} \pi[Z](r) &= \{s \mid \exists t \in r : (\exists t_1 \in t[Z_1] : s[\text{attr}(Z_1)] = t_1) \\ &\quad \wedge \ldots \wedge (\exists t_k \in t[Z_k] : s[\text{attr}(Z_k)] = t_k)\} \\ \text{format}(\pi[Z](R)) &= (Z_{1_1}\text{format}(Z_{1_1}), \ldots, Z_{1_l}\text{format}(Z_{1_l}), \cdots, \\ &\quad Z_{k_1}\text{format}(Z_{k_1}), \ldots, Z_{k_l}\text{format}(Z_{k_l})) \end{aligned}$$

The definition for case 2 has the following properties. If the projection list consists of one relation-valued attribute, the instances of the subrelation from all tuples form the result. Duplicate tuples are eliminated. If the projection list contains two or more relation-valued attributes, we first take the Cartesian product of their instances in the same tuple and then form the result relation as above (see Tables 10(a) and 10(b)).

As defined above, a projection does not always produce a result in PNF. This is illustrated in Tables 9 and 10. The relation instance shown in Table 9(a) is in PNF. Using the above definition of projection (case 1), we obtain the result shown in Table 9(b). The result is not in PNF, nor is it intuitively appealing. We would prefer the result of Table 9(c), which is in PNF. The same situation may occur when projecting onto relation-valued attributes



R

X_1	Y_1	Y_2	
	A	B	Y_3
			C
1	1	1	1
	2		2
			3
		2	1
			3
2	1	1	2
			4
		2	1
			4

(a)

$\pi[Y_1, Y_2](R)$

Y_1	Y_2	
A	B	Y_3
		C
1	1	1
		2
		3
1	2	1
		3
2	1	1
		2
		3
2	2	1
		3
1	1	2
		4
1	2	1
		4

(b)

$\pi[Y_1, Y_2](R)$

Y_1	Y_2	
A	B	Y_3
		C
1	1	1
		2
		3
		4
1	2	1
		3
		4
2	1	1
		2
		3
2	2	1
		3

(c)

Table 10: Projection and preservation of PNF for case 2.

alone (case 2). This is illustrated in Table 10. Again, the result in Table 10(b) is not in PNF, and we would prefer the result of Table 10(c), which is in PNF. We therefore redefine the effect of the projection operator so that the result is guaranteed to be in PNF. This definition of projection is used throughout the rest of the paper.

Projection with preservation of PNF: Again, there are two cases to consider:

1. $\text{sattr}(Z) \neq \emptyset$, that is, Z contains at least one single-valued attribute. Assume that $\text{rattr}(Z) = \{Z_1, Z_2, \ldots, Z_n\}$, $n < k$.

$$\pi[Z](r) = \{s \mid \exists t \in r : s[\text{sattr}(Z)] = t[\text{sattr}(Z)]$$
$$\wedge s[Z_1] = \{\bigcup_k t_k[Z_1] | \forall t_k \in r : s[\text{sattr}(Z)] = t_k[\text{sattr}(Z)]\}$$
$$\ldots$$
$$\wedge s[Z_n] = \{\bigcup_k t_k[Z_n] | \forall t_k \in r : s[\text{sattr}(Z)] = t_k[\text{sattr}(Z)]\}\}$$

2. sattr(Z) = $\emptyset$, that is, Z contains only relation-valued attributes. Assume that rattr(Z) = $\{Z_1, Z_2, \ldots, Z_k\}$; and assume that rattr(Z_i) = $\{Z_{i(1)}, Z_{i(2)}, \ldots, Z_{i(r_i)}\}$, $1 \leq i \leq k$, $r_i \geq 0$.

$$\pi[Z](r) = \{s | \exists t \in r : (\exists u_1 \in t[Z_1] : s[\text{sattr}(Z_1)] = u_1[\text{sattr}(Z_1)])$$
$$\wedge \ldots \wedge (\exists u_k \in t[Z_k] : s[\text{sattr}(Z_k)] = u_k[\text{sattr}(Z_k)])$$
$$\wedge s[Z_{1(1)}] = \{\bigcup_l t_l[Z_1][Z_{1(1)}] | \forall t_l \in r : \bigwedge_{j=1,k} s[\text{sattr}(Z_j)] = t_l[Z_j][\text{sattr}(Z_j)]\}$$
$$\ldots$$
$$\wedge s[Z_{1(r_1)}] = \{\bigcup_l t_l[Z_1][Z_{1(r_1)}] | \forall t_l \in r : \bigwedge_{j=1,k} s[\text{sattr}(Z_j)] = t_l[Z_j][\text{sattr}(Z_j)]\}$$
$$\ldots$$
$$\wedge s[Z_{k(1)}] = \{\bigcup_l t_l[Z_k][Z_{k(1)}] | \forall t_l \in r : \bigwedge_{j=1,k} s[\text{sattr}(Z_j)] = t_l[Z_j][\text{sattr}(Z_j)]\}$$
$$\ldots$$
$$\wedge s[Z_{k(r_k)}] = \{\bigcup_l t_l[Z_k][Z_{k(r_k)}] | \forall t_l \in r : \bigwedge_{j=1,k} s[\text{sattr}(Z_j)] = t_l[Z_j][\text{sattr}(Z_j)]\}\}$$

3.3 Selection

The selection operator of traditional relational algebra must be extended to be applicable to nested relations. What is required is the ability to make use of both the single-valued and relation-valued attributes of a relation in the selection. This is accomplished by allowing the selection predicate to contain relation algebra expressions over the relation-valued attributes of the relation, and also allowing set comparisons. The syntax of the selection operator is

$$\sigma[F](R)$$

where R is a relation expression and F is a selection predicate over attributes in scope(R). The important thing to note here is that although the selection predicate may be testing the tuples in a subrelation of R, *entire* tuples of R are selected. Like the set operators, selection is a format preserving operator, that is, the format of the result relation is the same as the format of the operand relation, R.

The selection predicate F is a logical combination of atomic selection conditions. We allow the standard boolean operators: *AND* ($\wedge$), *OR* ($\vee$) and *NOT* ($\neg$).

Definition 3.1 (Atomic selection condition) *An atomic selection condition has one of two forms: Z_1 op C or Z_1 op Z_2 where op $\in \{<, \leq, >, \geq, =, \neq, \subset, \subseteq, \supset, \supseteq\}$, C is either a single-valued or relation-valued constant, and Z_1, Z_2 are one of the following:*

- *the name of a single-valued or relation-valued attribute within the scope of the operand relation of the selection operation.*
- *a relational algebra expression operating on relation-valued attributes within the scope of the operand relation of the select statement.*

All scalar comparisons must be made between comparable domains. In set comparisons, the two operands must have equivalent formats.

The formal definition of the effect of the select operator at the instance level can now be given. Consider the selection statement

$$\sigma[F(Z)](R)$$

where $Z \subseteq \text{scope}(R)$ and F is a selection predicate over the attributes Z.

$$\begin{aligned} \sigma[F](r) &= \{s |\ \exists t \in r : s = t \wedge F(t[Z]) = \mathit{true}\} \\ \text{format}(\sigma[F(Z)](R)) &= \text{format}(R) \end{aligned}$$

Example 3.1 *Find every employee who has both a son and a daughter. (Refer to Table 1.)*

$$\text{Result} = \sigma[\pi[\text{SEX}](\text{CHILDREN}) = \{F, M\}](\text{EMPLOYEE})$$

When constructing selection conditions involving algebra expressions, the scoping rules of the operators must be strictly observed. Consider the relation R(A,B,C(D,E,F(G)),H(I,J)) and the selection

$$\sigma[\pi[\text{X}](\text{C}) = \emptyset](\text{R})$$

The question is: What attributes are allowed in the projection list X? The answer is provided by the scoping rules of the projection operator: the attributes in the projection list must be a subset of the attributes of the operand of the project. Hence, $\text{X} \subseteq \text{attr(C)} = \{\text{D,E,F,H}\}$

Now consider the selection

$$\sigma[\sigma[\text{A} > \text{D}](\text{C}) \neq \emptyset](\text{R})$$

Is the comparison A>D allowed? The answer is provided by the scoping rules of the selection operator: the selection predicate must be over attributes in the scope of the operand relation. The comparison is valid because C is in scope(R), and A and D are in scope(C). However, the scalar comparisons in

$$\sigma[\sigma[(\text{D} < \text{G}) \wedge (\text{B} = \text{I})](\text{C}) \neq \emptyset](\text{R})$$

are now allowed because G and I are not in scope(C).

3.4 Cartesian Product

Cartesian product can trivially be extended to tuples of nested relations. The syntax of the Cartesian product operator is

$$R_1 \times R_2$$

For the Cartesian product to be defined, we must have $\text{attr}(R_1) \cap \text{attr}(R_2) = \emptyset$. Let $\text{attr}(R_1) = (A_1, \ldots, A_m)$, and $\text{attr}(R_2) = (B_1, \ldots, B_n)$. The effect on the instance level is defined as follows:

$$\begin{aligned} r_1 \times r_2 &= \{s |\ \exists t_1 \in r_1, \exists t_2 \in r_2 :\ s[\text{attr}(R_1)] = t_1 \wedge s[\text{attr}(R_2)] = t_2\} \\ \text{format}(R_1 \times R_2) &= (A_1\text{format}(A_1), \ldots, A_m\text{format}(A_m), \\ &\qquad B_1\text{format}(B_1), \ldots, B_n\text{format}(B_n)) \end{aligned}$$

3.5 Join and Natural Join

In traditional relational algebra, a join is simply a Cartesian product followed by a selection. We retain this feature and define the join as

$$R_1 \bowtie [F]\ R_2 \equiv \sigma[F](R_1 \times R_2)$$

The *natural join* consists of the join without the selection predicate F. The implied selection predicate is equality between the common attributes, that is, attributes in R_1 and R_2 with the same name. If a common attribute is relation-valued, set equality is implied. As in traditional relational algebra, only one copy of each common attribute is retained. The syntax is:

$$R_1 \bowtie R_2$$

where R_1, R_2 are relational expressions. Let $\text{attr}(R_1) = \text{VZ}$ and $\text{attr}(R_2) = \text{WZ}$, where $\text{V} = \{V_1, \ldots, V_k\}$, $W = \{W_1, \ldots, W_l\}$, and $\text{Z} = \{Z_1, \ldots, Z_n\}$ are sets of attributes. Then $\text{attr}(R_1) \cap \text{attr}(R_2) = \text{Z}$ is the set of common attributes.

$$\begin{aligned} r_1 \bowtie r_2 &= \{s | \exists t_1 \in r_1, \exists t_2 \in r_2 : s[V] = t_1[V] \wedge s[\text{W}] = t_2[\text{W}] \\ &\qquad \wedge s[\text{Z}] = t_1[\text{Z}] \wedge t_1[\text{Z}] = t_2[\text{Z}]\} \\ \text{format}(R_1 \bowtie R_2) &= (V_1\text{format}(V_1), \ldots, V_k\text{format}(V_k), Z_1\text{format}(Z_1), \\ &\qquad \ldots, Z_n\text{format}(Z_n), W_1\text{format}(W_1), \ldots, W_l\text{format}(W_l)) \end{aligned}$$

An example of the natural join is given in Table 11. If a natural join is applied to a subrelation, two attributes are considered to have the same name if they agree on the *last* component of the complete attribute name, that is, the pathname is disregarded.

π[CNO,CNAME,TERM,ENROLLMENT](OFFERINGS $\bowtie$ COURSES)

CNO	CNAME	TERM	ENROLLMENT	
			SNO	GRADE
CS 240	Prog. Principles	F86	8653	85
			8621	80
CS 340	Data Structures	W87	8621	78
			8653	82
			8619	90
CS 448	Dabatase Mgmt.	S87	8621	83
			8653	88
			8619	92
CO 230	Combinatorics	W86	8380	67
			8470	93
CO 230	Combinatorics	F86	8653	87
			8619	82
CO 330	Enumeration	W87	8621	47
			8619	92
			8653	80

Table 11: List the course number, name, term and enrollment of the courses offered.

Example 3.2 *Find the course number of every course taken by both D. Walters and A. Brown.*

$$\begin{aligned}\text{Result} \;=\; & \pi[\text{CNO}]\sigma[\pi[\text{NAME}](\text{ENROLLMENT} \bowtie \text{STUDENT}) \\ & \supseteq \{\text{'D. Walters'},\text{'A.Brown'}\}](\text{OFFERINGS})\end{aligned}$$

Clearly, we want the join ENROLLMENT $\bowtie$ STUDENT to be over attribute SNO. However, the full attribute name of SNO in ENROLLMENT is ENROLLMENT.SNO. Hence, the above rule.

3.6 Nest

The nest operator creates a new subrelation and thus changes the structure of a relation. It was first introduced by Jaeschke and Schek in [JS82]. It is not found in 1NF algebra. The syntax is

$$\nu[Z' = (Z)](R)$$

where R is a relational algebra expression, $Z \subset \text{attr}(R)$ is the list of attributes over which to nest, and Z' is the new name given to the the subrelation composed of the nested attributes. We now state formally what happens

at the instance level. Let $V = \{V_1, \ldots, V_m\}$, $Z = \{Z_1, \ldots, Z_n\}$, $m, n \geq 1$; let $\text{attr}(R) = \{V, Z\}$. It is required that $\text{sattr}(V) \neq \emptyset$, and $\text{sattr}(Z) = \emptyset$.

$$\begin{aligned} \nu[Z' = (Z)](r) &= \{s \mid \exists t \in r : s[V] = t[V] \\ &\quad \wedge s[Z'] = \{u[Z] \mid \exists u \in r : u[V] = s[V]\}\} \\ \text{format}(\nu[Z' = (Z)](R)) &= (V_1\text{format}(V_1), \ldots, V_m\text{format}(V_m), Z'(Z_1\text{format}(Z_1), \\ &\quad \ldots, Z_n\text{format}(Z_n))) \end{aligned}$$

An example using the nest operator is given in Table 12.

3.7 Unnest

Unnest is the inverse of nest. This operator was also introduced by Jaeschke and Schek in [JS82]. It is not found in 1NF algebra. The syntax is

$$\mu[Y](R)$$

where $Y \in \text{attr}(R)$ is a relation-valued attribute. For this operator to be defined, we must have $\text{attr}(Y) \cap (\text{attr}(R)\text{-}\{Y\}) = \emptyset$. The subrelation specified by Y is unnested. We now specify what happens at the instance level. Let $\text{attr}(Y) = \{Y_1, \ldots, Y_n\}$, and $\text{attr}(R) = \{Z_1, \ldots, Z_m, Y\}$.

$$\begin{aligned} \mu[Y](r) &= \{s \mid \exists t \in r : s[\text{attr}(R) - Y] = t[\text{attr}(R) - Y] \\ &\quad \wedge s[\text{attr}(Y)] \in t[Y]\} \\ \text{format}(\mu[Y](R)) &= (Z_1\text{format}(Z_1), \ldots, Z_m\text{format}(Z_m), Y_1\text{format}(Y_1), \\ &\quad \ldots, Y_n\text{format}(Y_n)) \end{aligned}$$

Note that the unnest operator is *not* information preserving. The unnesting of empty subrelations results in null values, which are not allowed in this algebra. To avoid this, the tuples which would have contained the null values after unnesting are discarded, resulting in loss of information.

3.8 The Rename Operator

This is an operator used to rename attributes. The problem of duplicate names in a relation can arise, for example, when taking a Cartesian product. To avoid this, one of the duplicate pair of attributes must be renamed. The operator is defined as:

$$\delta[Z \leftarrow Z'](R)$$

where $Z \in \text{attr}(R)$, and Z' is a new attribute name such that $Z' \notin \text{attr}(R)$. We now specify what happens at the instance level. Let $\text{attr}(R) = \{Z_1, \ldots, Z_n, Z\}$.

$$\delta[Z \leftarrow Z'](r) =$$

ν[STUBYFACULTY = (SNO, NAME)](STUDENT)

FAC	STUBYFACULTY	
	SNO	NAME
Science	8380	D. Walters
Mathematics	8470	A. Brown
	8619	T. Lonsdale
	8621	R. Helbig
	8653	G. Wright

Table 12: An example using the Nest operator.

$$\{s|\ \exists t \in r : s[\text{attr}(R) - Z] = t[\text{attr}(R) - Z] \wedge s[Z'] = t[Z]\}$$

$$\text{format}(\delta[Z \leftarrow Z'](R)) \quad = \quad (Z_1\text{format}(Z_1), \ldots, Z_n\text{format}(Z_n), Z'\text{format}(Z))$$

As a convenient shorthand, we define

$$\delta[Z_1 \leftarrow Z_1', Z_2 \leftarrow Z_2', \ldots, Z_k \leftarrow Z_k'](R) =$$
$$\delta[Z_K \leftarrow Z_k'](\ldots(\delta[Z_2 \leftarrow Z_2'](\delta[Z_1 \leftarrow Z_1'](R)))\ldots)$$

4 The Subrelation Constructor

The operators defined in the previous sections only allow us to combine and modify relations at the root. This is clearly not sufficient; we also need to be able to modify the interior of a nested relation. In this section we introduce the concept of a *subrelation constructor* which provides this capability. As an introduction, consider the following query against the university database.

Example 4.1 *For each course offered, list the student number, student name and grade of all students who received a grade of 85 or above.*

To answer this query we must modify the subrelation ENROLLMENT by selecting tuples where GRADE $\geq$ 85 and then joining with the STUDENT relation to obtain the student name. Using our new subrelation constructor this query can be expressed as follows:

R := $\{$(CNO,TERM,SCHOLARS); SCHOLARS := σ[GRADE $\geq$ 85]
(ENROLLMENT) $\bowtie$ π[SNO,NAME](STUDENT)$\}$(OFFERINGS)

This expression is interpreted in the following way. For each tuple (at the root level) of OFFERINGS, construct a new tuple which consists of CNO, TERM and a new subrelation SCHOLARS. (The subrelation ENROLLMENT disappears.) SCHOLARS is constructed from the tuple's ENROLLMENT relation and the (external) relation STUDENT, as specified by the given algebra expression. The result is shown in Table 13.

CNO	TERM	SCHOLARS		
		SNO	NAME	GRADE
CS 240	F86	8653	G. Wright	85
CS 340	W87	8619	T. Lonsdale	90
CS 448	S87	8653	G. Wright	88
		8619	T. Lonsdale	92
CS 488	W88			
CO 230	W86	8470	A. Brown	93
CO 230	F86	8653	G. Wright	92
CO 330	W87	8619	T. Lonsdale	92

Table 13: For each course offered, select students with a grade of 85 or above.

The syntax of the subrelation constructor is

$$\{P(A_1, \ldots, A_k); A_{i_1} := E_1, \ldots, A_{i_l} := E_l\}(R)$$

where P is a pathname in relation R, $A_1, \ldots, A_k$ are attribute names and $E_1, \ldots, E_l$ are relational algebra expressions. Each expression E_j specifies how the value of the attribute A_{i_j} is to be computed. If an attribute in the attribute list does not occur in the expression list, then it retains its old value. An expression E_j can only operate on relations in scope(P).

We define the effect of the subrelation constructor for the case when only one new subrelation is constructed. The extension to the case of constructing several new subrelations is conceptually straightforward, though unwieldy. There are two cases to consider:

1. P is empty. We consider the constructor

$$\{(Z, W); W := E(V_0, R_2)\}(R_1)$$

where $Z = (Z_1, \ldots, Z_n) \subseteq \text{attr}(R_1)$ are the attributes to be preserved; W is the new subrelation to be computed; $V_0 \subseteq \text{rattr}(R_1)$ are relation-valued attributes of R_1; and R_2 is some relation in the database. The effect of this operation is then

$$\{(Z, W); W := E(V_0, R_2)\}(r_1) =$$
$$\{s | \exists t \in r_1 : s[Z] = t[Z] \wedge s[W] = E(t[V_0], r_2)\}$$
$$\text{format}(\{(Z, W);\ W := E(V_0, R_2)\}(R_1)) =$$
$$(Z_1\text{format}(Z_1), \ldots, Z_n\text{format}(Z_n), W\text{format}(E(V_0, R_2)))$$

Note that $t[V_0]$ represents a set of relation instances, and r_2 a relation instance.

2. P is not empty, $P = Y_1.Y_2.\cdots.Y_k$. The constructing expression may now operate on any relation-valued attributes in scope(P). We consider the constructor

$$\{Y_1.Y_2.\cdots.Y_k(Z,W);\ W := E(V_0,V_1,\ldots,V_k,R_2)\}(R_1)$$

where $Z \subseteq$ iscope(P) are the attributes to be preserved; W is the new subrelation to be computed; $V_0 \subseteq$ rattr(R_1); $V_j \subseteq$ rattr($Y_1.\cdots.Y_j$) ($1 \leq j \leq k$); and R_2 is some relation in the database. The effect of this operator is then

$\{Y_1.Y_2.\cdots.Y_k(Z,W);\ W := E(V_0,\ldots,V_k,R_2)\}(r_1) =$
$\quad \{s \mid \exists t \in r_1 : s[\text{attr}(R) - Y_1] = t[\text{attr}(R) - Y_1]$
$\quad \wedge s[Y_1] = \{Y_2.Y_3.\cdots.Y_k(Z,W);\ W := E(t[V_0],V_1,\ldots,V_k,R_2)\}(t[Y_1])\}$
format($\{Y_1.Y_2.\cdots.Y_k(Z,W);\ W := E(V_0,\ldots,V_k,R_2)\}(R_1)$) =
$\quad$ format(R_1) except that the term Y_1 format(Y_1) is replaced by
$\quad Y_1$format($\{Y_2.Y_3.\cdots.Y_k(Z,W);\ W := E(V_0,\ldots,V_k,R_2)\}(Y_1)$)

The algebra expressions used within a subrelation constructor may, of course, in turn contain subrelation constructors. However, this is needed only rarely. The subrelation constructor is a simple, but very powerful operator, as the following examples show.

Example 4.2 *For every course offering in Fall 86, list course number and student number, name and grade of every student enrolled in the course.*

T1 := σ[TERM='F86'](OFFERINGS)
Result := {(CNO,ENROLLED); ENROLLED:=
ENROLLMENT $\bowtie$ π[SNO,NAME](STUDENT)}(T1)

Example 4.3 *For all faculties and all departments, list every course offering, giving course number and term, where no student recieved a grade of 90 or higher.*

T1 := π[CNO,TERM]
$\sigma[\sigma$[GRADE $\geq$ 90](ENROLLMENT) = $\emptyset$](OFFERINGS)
Result := {DEPARTMENT(DNAME,DC);
DC:=DEPARTMENT.COURSES $\bowtie$ $T1$}(FACULTY)

SNO	SNAME	REM
		CNO
8619	T. Lonsdale	CS 240
8621	R. Helbig	CO 230

Table 14:

Example 4.4 *For each course, list the course number, name and the course number and name of its prerequisite courses.*

$$
\begin{aligned}
\text{T1} &:= \text{COURSES} \\
\text{Result} &:= \{(\text{CNO,CNAME,REQ}); \\
&\qquad \text{REQ} := \pi[\text{CNO,CNAME}](\text{PREREQ} \bowtie \text{T1})\}(\text{COURSES})
\end{aligned}
$$

Example 4.5 *For each student who has not taken all the prerequisite courses to the courses he/she has taken, list student number, name and the missing courses. The query is posed below, and the result is shown in Table 14.*

$$
\begin{aligned}
\text{T1} &:= \nu[\text{CT=(CNO)}]\pi[\text{CNO,SNO}]\mu[\text{ENROLLMENT}](\text{OFFERINGS}) \\
\text{T2} &:= \{(\text{SNO,REM}); \text{REM} := \pi[\text{PREREQ}](\text{CT} \bowtie \text{COURSES}) - \text{CT}\}(T1) \\
\text{Result} &:= (\pi[\text{SNO,SNAME}](\text{STUDENT})) \bowtie (\sigma[\text{REM} \neq \emptyset](\text{T2}))
\end{aligned}
$$

5 Comparisons

In this section, comparisons are made between our algebra and the algebra proposed by [SS86], which we call the Schek and Scholl algebra.

Example 5.1 *This query is against the relation* EMP *in Table 1. Find all employees having a son or daughter with the same name, listing employee number, name and date of birth of the child.*

- Schek and Scholl algebra

$$
\begin{aligned}
\text{T1} &:= \pi[\text{ENO,NAME,} \\
&\qquad (\pi[\text{NAME:CNAME,DOB}](\text{CHILDREN})):\text{JUNIOR}](\text{EMP}) \\
\text{T2} &:= \pi[\text{ENO,NAME,}(\pi[\text{DOB}](\sigma[\text{CNAME=NAME}](\text{JUNIOR}))):\text{JR}](\text{T1}) \\
\text{Result} &:= \mu[\text{JR}](\text{T2})
\end{aligned}
$$

- Our algebra

$$\begin{aligned}\text{Result} \;:=\; & \nu[JR]\{(\text{ENO,NAME,JR});\ \text{JR} := \pi[\text{DOB}] \\ & \sigma[\text{CHILDREN.NAME} = \text{NAME}]\ (\text{CHILDREN})\}\ (\text{EMP})\end{aligned}$$

The following queries are against the example relation used in [SS86]. The scheme is as follows: DEPT(D, DN, AE(AN,AJD), TE(TN,TJD,C(CN,Y))). AE and TE contain information about administrative and technical employees, respectively. AJD and TJD are job descriptions and C contains information about courses taken by an employee.

Example 5.2 *(Q7 in [SS86]) Find the technical and administrative employees for each department who share the same job description.*

- Schek and Scholl algebra

$$\begin{aligned}\text{Result} \;:=\; & \pi[\text{D,DN,}(\pi[\text{AN,TN,TJD}] \\ & (\sigma[\text{AJD=TJD}](\text{AE}\times\text{TE}))):\text{ATE}](\text{DEPT})\end{aligned}$$

- Our algebra

$$\begin{aligned}\text{Result} \;:=\; & \{(\text{D,DN,ATE});\text{ATE:=} \\ & \pi[\text{AN,TN,TJD}](\text{AE}\bowtie[\text{AJD=TJD}]\ \text{TE})\}(\text{DEPT})\end{aligned}$$

Example 5.3 *(Q8 in [SS86]) Assume that we have a second relation* CD(CN,CDES) *containing course descriptions. Change the relation* DEPT *to include the names of the courses which each employee has taken.*

- Schek and Scholl algebra

$$\begin{aligned}\text{S1} \;:=\; & \pi[\text{CN:C\#,CDES}](\text{CD}) \\ \text{Result} \;:=\; & \pi[\text{D,DN,AE,}\pi[\text{TN,TJD,}\pi[\text{CN,CDES,Y}] \\ & (\sigma[\text{CN=C\#}](\text{C}\times\text{S1})) : \text{TCD}](\text{TE}) : \text{TEC}](\text{DEPT})\end{aligned}$$

- Our algebra

$$\text{Result} \;:=\; \{\text{TE(TN,TJD,TCD)};\ \text{TCD} := \text{C}\bowtie\text{CD})\}(\text{DEPT})$$

The following four queries are the same as the last four queries of the previous section. The queries are posed in the algebra of [SS86].

Example 5.4 *For every course offering in Fall 86, list course number and student number, name and grade of every student enrolled in the course.*

$$
\begin{aligned}
S1 &:= \sigma[\text{TERM='F86'}](\text{OFFERINGS}) \\
S2 &:= \pi[\text{SNO:S\#,NAME}](\text{STUDENT}) \\
\text{Result} &:= \pi[\text{CNO},\pi[\text{SNO,NAME,GRADE}](\sigma[\text{S\#=SNO}] \\
&\quad (\text{ENROLLMENT} \times \text{S2})) : \text{ENROLLED}](S1)
\end{aligned}
$$

Example 5.5 *For all faculties and all departments, list every course and course offering, giving course number and term, where no student received a grade of 90 or higher.*

$$
\begin{aligned}
\text{S1} &:= \pi[\text{CNO:C\#,TERM}] \\
&\quad \sigma[\sigma[\text{GRADE} \geq 90](\text{ENROLLMENT}) \neq \emptyset](\text{OFFERINGS}) \\
\text{Result} &:= \pi[\text{FNAME},\pi[\text{DNAME},(\pi[\text{CNO,TERM}](\sigma[\text{CNO=C\#}] \\
&\quad (\text{COURSES} \times \text{S1}))):\text{DC}](\text{DEPARTMENT}):\text{DEPT}](\text{FACULTY})
\end{aligned}
$$

Example 5.6 *For each course, list the course number, name and the course number and names of the prerequisite courses.*

$$
\begin{aligned}
\text{S1} &:= \pi[\text{CNO:C\#,CNAME}](\text{COURSES}) \\
\text{Result} &:= \pi[\text{CNO,CNAME},(\pi[\text{CNO,CNAME}](\sigma[\text{CNO=C\#}] \\
&\quad (\text{PREREQ} \times \text{S1}))):\text{REQCOURSES}](\text{COURSES})
\end{aligned}
$$

Example 5.7 *For each student who has not taken all the prerequisite courses to the courses he/she has taken, list student number, name and the missing courses.*

$$
\begin{aligned}
\text{S1} &:= \nu[\text{CNO:CTAKEN}]\pi[\text{CNO,SNO}]\mu[\text{ENROLLMENT}](\text{OFFERINGS}) \\
\text{S3} &:= \pi[\text{CNO:C\#}, \pi[\text{CNO:C\#}](\text{PREREQ}):\text{REQD}](\text{COURSES}) \\
\text{S4} &:= \pi[\text{SNO},\pi[\text{CNO,REQD}](\sigma[\text{C\#=CNO}]\ (\text{CTAKEN} \times \text{S3})):\text{CT}](\text{S1}) \\
\text{S5} &:= \pi[\text{SNO:S\#},\pi[\text{C\#}](\mu[\text{PREREQ}](\text{CT})):\text{CREQD}](\text{S4}) \\
\text{S6} &:= \pi[\text{SNO,CREQD,CTAKEN}](\sigma[\text{SNO=S\#}](\text{S1} \times \text{S5})) \\
\text{S7} &:= \pi[\text{SNO,CREQD},\pi[\text{C\#:CNO}](\text{CTAKEN}):\text{CTAKE}](\text{S6}) \\
\text{S8} &:= \pi[\text{SNO},(\text{CREQD-CTAKEN}):\text{CMISSING}](\text{S7}) \\
\text{S9} &:= \pi[\text{SNO:S\#,NAME,FAC}](\text{STUDENT}) \\
\text{Result} &:= \pi[\text{SNO,NAME,CMISSING}]\sigma[\text{SNO=S\#}](\text{S8} \times \text{S9})
\end{aligned}
$$

6 Discussion

We have in this paper defined an algebra for nested relations. Our objective was to design an algebra which achieves a balance between power and ease of use and understanding. Our design was strongly influenced by the algebra of Schek and Scholl [SS86]. The main difference is in the way modification of subrelations is handled. In addition, we also require that all relations be in PNF. We feel that our concept of a subrelation constructor is easier to understand and leads to simpler queries than the algebra of Schek and Scholl.

Our algebra is more restricted than the algebra for Verso relations designed by Abiteboul and Bidoit [AB84a]. For example, we chose not to allow set operators on relations with different formats, which is allowed in the Verso algebra. This was dictated by our desire to keep the algebra conceptually simple.

We plan to use our algebra in a new database system based on nested relations. It will be used internally in the system to specify queries and query execution plans. (The external query language will be different.) To do so, the algebra must be further extended to allow, for example, arithmetic expressions and aggregate functions (count,max,sum,avg). This can easily be done without changing the basic structure of the algebra.

References

[AB84a] S. Abiteboul and N. Bidoit, Non First Normal Form Relations: An Algebra Allowing Data Restructuring, *Technical Report, No 347*, INRIA, France, 1984.

[AB84b] S. Abiteboul and N. Bidoit, Non First Normal Form Relations to Represent Hierarchically Organized Data, *2nd ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, 1984, p. 191-200.

[FK77] A.L. Furtado and L. Kerschberg, An Algebra for Quotient Relations, *Proc. ACM SIGMOD Conference of Management of Data*, 1977, p. 1-8.

[FT83] P.C. Fischer and S.J. Thomas, Operators for Non-First-Normal-Form Relations, *Proc. IEEE COMPSAC* 1983, p. 464-475.

[JS82] G. Jaeschke and H.-J. Schek, Remarks on the algebra of non-first-normal-form relations, *Proc. ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, 1982, p. 124-138.

[M77] A. Makinouchi, A Consideration on Normal Form of Not-Necessarily-Normalized Normalized Relation in the Relational Data Model, *Proc. of the Third International International Conference on Very Large Data Bases*, Tokyo (October 1977), p. 447-453.

[OO83] Z. M. Ozsoyoglu and G. Ozsoyoglu, An Extension of Relational Algebra for Summary Tables, *Proc. of the Second International Workshop on Statistical Database Management*, 1983, p. 202-211.

[OY85] Z.M. Ozsoyoglu and L. Yuan, A Normal Form for Nested Relations, *Proc. of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Portland, March, 1985 p. 251-260.

[OY87] Z.M. Ozsoyoglu and L. Yuan, A New Normal Form for Nested Relations, *ACM Transactions on Database Systems*, March, 1987, p. 111-136.

[RK87] M.A. Roth and H.F.Korth, The Design of ¬1NF Relational Databases into Nested Normal Form, *Proc. of ACM/SIGMOD Annual Conference*, San Francisco, 1987, p. 143-159.

[RKB85] M.A. Roth, H.F. Korth and D.S. Batory, SQL/NF: A Query Language for ¬1NF Relational Databases, *Technical Report, TR-85-19*, University of Texas at Austin, 1985.

[RKS84] M. A. Roth and H. F. Korth and A. Silberschatz, Extended Algebra and Calculus for ¬1NF Relational Databases, *Technical Report, TR-84-36*, University of Texas at Austin, 1984.

[SS86] H.-J. Schek and M.H. Scholl, The Relational Model with Relation-Valued Attributes, *Information Systems*, Volume 11, No. 2, 1986, p. 137-147.

[TF86] Stan J. Thomas and Patrick C. Fischer, Nested Relational Structures, *Advances in Computing Research*, Vol. 3, 1986, JAI Press Inc., p. 269-307.