PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 10/76

AN ALGEBRA OF QUOTIENT RELATIONS
(DRAFT)

by

A. L. Furtado

and

L. Kerschberg

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

Series: Monographs in Computer Science
and Computer Applications

Nº 10/76

AN ALGEBRA OF QUOTIENT RELATIONS*
(DRAFT)

by

A. L. Furtado

and

L. Kerschberg+

Series Editor: Roberto Lins de Carvalho September 1976

* This research has been sponsored in part by FINEP, under contract nº 370/CT

+ Present address: Department of Information Systems Management, University of Maryland, College Park, Md., 20742

ERRATA

page	line	Correction
7	22	$R' \leftarrow R' * (A-C)$
8	1	$R' \leftarrow R' * (A-B)$
8	2	$S' \leftarrow S' * (B-A)$
12	12	$F \leftarrow (\hat{D} \circledast \hat{E})[(N_1,T_1,L_1) = (N_2,T_2,L_2)]$
13	6	$(\hat{R} \circledast \hat{S})\ [\underline{A} \ominus \underline{B}] * I_{1,2}$
15	19	(except when indispensable)
18	7	$Y \leftarrow \overset{\vee}{R}[T,L,O]$
18	11	$B \leftarrow ((Y/(T,L) \circledast \overset{\vee}{X}/L)[(L_1,O_1) \supseteq (L_2,O_2)])[T,L_1]$
19	4	$Q \leftarrow ((\overset{\vee}{Z}/B \circledast \overset{\vee}{D}/C)[\underline{A_1} \supseteq \underline{A_2}])[\underline{E}]$
19	6	$\overset{\vee}{Z} \leftarrow (((Q \circledast D)[C_1 = C_2][\underline{C_2}]) \circledast R) * I$

For copies, c

Rosane Teles L
Head, Setor de
Depto. de Informatica - PUC/RJ
Rua Marques de São Vicente, 209 - Gávea
20.000 - Rio de Janeiro - RJ - BRASIL

ABSTRACT:

An algebra which operates on partitioned relations is described. It is shown that the proposed algebra is as powerful as the original relational algebra, having the advantage of greater flexibility for expressing queries in ways that are more straightforward and conducive to a more efficient processing. The convenience of using the algebra as an intermediate language is indicated.

KEY WORDS:

Data bases, relational model, relational algebra, partitioning.

RESUMO:

Uma algebra que opera sobre relações particionadas é descrita. Mostra-se que a algebra proposta é tão poderosa como a algebra relacional original, tendo maior flexibilidade para expressar consultas de modo mais direto e conducente a um processamento mais eficiente. A conveniência de usar a algebra como linguagem intermediária é indicada.

PALAVRAS CHAVE:

Bancos de dados, modelo relacional, algebra relacional, partições.

CONTENTS

1. INTRODUCTION

In recent years a large number of data models have been proposed and developed. These models consist of an information structure together with some sort of algebra (data manipulation language) to process the information.

Historically, the first formal data model was CODASYL's Information Algebra [1], proposed in 1962 and later extended by Kobayashi [2]. In 1968, Childs proposed his Extended Set Theory which was another conceptual breakthrough [3]. Childs recognized that pure set theory was insufficient to describe n-tuples because of the problems associated with their representation in the computer. The model which perhaps has generated the most interest and research is Codd's Relational Model [4], together with its relational algebra and calculus [5].

A more complete overview of data models, their differences and their similarities, may be found in [6].

The Information Algebra and Extended Set Theory models are quite general in that the user obtains information by creating areas and sets, respectively. The Relational Model, while also being general in providing the n-ary relations as the information structure, offers the added benefit of a "conceptual schema" consisting of relation names and domain names. Recently ANSI/SPARC [7] has proposed three schema levels for a data base management system (DBMS): the external, conceptual, and internal schema levels.

It is the authors' opinion that a conceptual schema such as the relational schema is important to establish a consensus as to the information content of a database. Further, it would be most desirable to endow the relational model with a more explicit set processing capability; this idea is present in high-level languages such as SEQUEL [8] (the "grouped by" option) and QUEL [9] ("aggregates").

This paper deals with an algebra of quotient relations which does precisely that. The basic idea is to define equivalence relations on n-ary

relations. Each such equivalence relation on an n-ary relation partitions it into a quotient relation consisting of equivalence classes. The algebra includes unary and binary operators which take the quotient relations as arguments, process equivalence classes as units, and yield another quotient relation as the result.

It will be shown that the algebra of quotient relations allows the user to formulate his queries in a more set oriented fashion than the original relational algebra.

2. PRESENTATION OF THE ALGEBRA

Let R be an n-ary relation and A a set of its attibutes*. If two tuples t and t' of R have the same values with respect to the attributes in A we can write

$$t \equiv_A t'$$

and note that the mathematical relation "$\equiv_A$" is an equivalence relation.

An equivalence relation partitions a set (in this case R, as a set of tuples) into disjoint blocks . A partitioned relation will be called a quotient relation.

For different choices of A , different quotient relations can be obtained. Together they constitute the family of quotient relations over R, Q_R.

It can be readily verified that Q_R is a finite lattice, with universal bounds the quotient relation consisting of a single block $(\check{R})$, and the quotient relation where each tuple is a block $(\hat{R})$. The partial order in the lattice is given by the degree of refinement. Figure 1 shows Q_R for $R \subseteq A \times B \times C$.

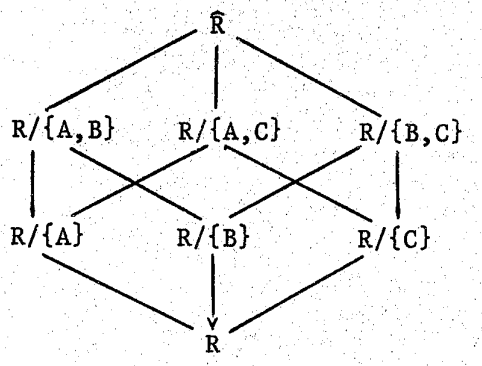

Fig. 1: A family of quotient relations

* Here an attribute is a domain playing a specific role in a relation.

In the Hasse diagram in figure 1, R/{A,B}is "above" R/{A},which indicates that one block of the latter may correspond to several blocks of the former.

The present algebra will use quotient relations exclusively. We assume that R, which is isomorphic to $\overset{\vee}{R}$, is given initially; other members of Q_R will be obtained by the application of the partitioning operator, "/", whose effect can be reversed by the de-partitioning operator, "*".

Let ϕ be the empty set, I the set of all attributes in R, A and B any two sets of such attributes, and R' a member of Q_R. The behavior of the operators above must obey the following requirements:

$$(R'/A)/B =_{def} (R'/B)/A =_{def} R'/(A \cup B)$$
$$(R'/A)*B =_{def} R'/(A - B)$$
$$\overset{\vee}{R} =_{def} \overset{\vee}{R}/\phi$$
$$\hat{R} =_{def} \overset{\vee}{R}/I$$

And since, in addition, we have

$$\phi - A =_{def} \phi$$
$$I \cup A =_{def} I$$
$$I - A =_{def} \bar{A}$$

it follows, for example, that

$$R' * I = \overset{\vee}{R}$$
$$R'/I = \hat{R}$$
$$R' * \phi = R'$$
$$R' / \phi = R'$$
$$R' * A = \overset{\vee}{R}/\bar{A}$$

We now introduce the relations to be used in the examples: R(N,T,O,L), S(N,C,S,T,M), W(N,C,S,T,M), where:

N - employee name
T - team
O - operation
L - location
C - category (of employee)
S - salary
M - manager name

Table 1 shows $\overset{\vee}{R}$, $\overset{\vee}{S}$ and $\overset{\vee}{W}$, in tabular representation (which is identical to the relational representation for R,S and W):

$\overset{\vee}{R}$

N	T	O	L
b	1	7	x
a	1	3	x
a	1	12	y
a	1	7	x
c	2	3	x
c	2	12	y
d	2	3	x
e	2	3	x

$\overset{\vee}{S}$

N	C	S	T	M
a	1	20	1	b
b	2	30	1	u
c	1	25	2	d
d	2	20	2	u
e	2	20	2	d

$\overset{\vee}{W}$

N	C	S	T	M
f	1	23	1	b
g	1	15	1	b
h	1	20	2	d
i	1	18	2	d
j	1	15	2	d

Table 1: Tabular representation of $\overset{\vee}{R}$, $\overset{\vee}{S}$, $\overset{\vee}{W}$

Table 2 shows $\overset{\vee}{R}$ partitioned by team and location: $Z \leftarrow \overset{\vee}{R}/\{T,L\}$. The "flat file" representation is maintained*.

* As it stands, the present algebra does not support a hierarchical approach - recall that we partition by sets rather than by sequences of attributes. An implementation may provide a hierarchical format for output (cf. section 4), but this has nothing to do with the algebra itself.

Z

N	T	O	L
b	1	7	x
a	1	3	x
a	1	7	x
a	1	12	y
c	2	3	x
d	2	3	x
e	2	3	x
c	2	12	y

Table 2: Example of partitioning

The other operators of the algebra are projection, restriction, Cartesian product, and union. Since their counterparts have been defined in the relational algebra [5], all we have to do is to show in what aspects they differ.

Firstly, as it would be expected, all operators are applied to quotient relations and yield quotient relations as result. Consider two quotient relations R' partitioned by A, and S' partitioned by B; let C and D be other sets of atributes in R', whereas $\underline{C}$ and $\underline{D}$ are sequences of the same attributes. Then the results are partitioned as follows:

- projection, $T \leftarrow R'[\underline{C}]$ - the operation can only take place if $A \subseteq C$; T is still partitioned by A;

- restriction, $T \leftarrow R'[\underline{C} \ominus \underline{D}]$ - T (a subset of R') is still partitioned by A;

- Cartesian product, $T \leftarrow R' \otimes S'$ - T is partitioned by $A \cup B$, which in fact contains all attributes in A and all attributes in B because, for this operation, the attributes of R' and S' are regarded as distinct (and are all kept in T);

- union, $T \leftarrow R' \oplus S'$ - the operation can only take place if $A = B$; here the attributes in R' and S' are required to be pairwise compatible [5]; T is partitioned by A (or, equivalently, by B)

Blocks are treated as units by the operators. Projection preserves the blocks, simply removing certain attributes (which may cause the elimination of duplicate tuples inside some blocks); restriction effects the acceptance or rejection of entire blocks; Cartesian product creates one block for each pair of blocks in the operands (containing the concatenated pairs of tuples); union merges the pairs of blocks from the operands that are equivalent under the common partitioning scheme and keeps also the unpaired blocks from each operand.*

Accordingly, the θ comparison operators used in restriction operate on sets (possibly singletons); θ may be one of $\{=,\tilde{=},\supseteq,\supset,\subseteq,\subset,\geq,>,\leq,<\}$, which may be modified by an overwritten "/" (negation) or, in the case of the four last ones, by "." (to be explained later). The operator "$\tilde{=}$" means that the two sets being compared have at least one element in common† (hence "$\not\tilde{=}$" means disjoint).

It is an undesirable burden to a person writing algebra expressions to keep track of the partitioning scheme of intermediate results.§ The following conventions are introduced in order to avoid this need:

a. For projection, if $A \not\subseteq C$, the implicit operation $R' \leftarrow R'*(A-\bar{C})$ will be assumed to precede the projection (i.e. the operand is partitioned by $A \cap C$).

---

* cf. the concept of congruence in [10] page 51.

† For two sets S_1 and S_2, $S_1 \tilde{=} S_2$ means $(\exists s_1 \in S_1, \exists s_2 \in S_2) s_1 = s_2$; the reader will note that the six first operators are expressible by quantification schemes with equality.

§ By contrast, if algebra expressions are produced by a translating program from a higher language, an optimizer routine could profitably take into consideration the partitioning schemes.

b. For union, if $A \neq B$, the implicit operations $R' \leftarrow R' * (A-\bar{B})$ and $S' \leftarrow S' * (B-\bar{A})$ will be assumed to precede the union (i.e. both operands are partitioned by $A \cap B$).

c. Whenever $\breve{R}$ appears in an expression it will be assumed to be available, noting that it can always be obtained from R' by taking $R' * A$ or, if A is not known, by $R' * I$; the availability of $\hat{R}$ is also assumed, for analogous reasons.

To avoid ambiguity when attributes with the same name appear in a resulting relation, subscripts will be used. Braces will be omitted in the case of partitioning by singleton sets.

A number of examples illustrate the use of the algebra to process queries.

Q_1: find the pairs of teams and locations, such that the team has participated in all operations taking place at the location; together with each team give also all its employees.*

operations	comments
$A \leftarrow \breve{R}[N,T,O] / T$	operations grouped by team
$B \leftarrow \breve{R}[O,L] / L$	operations grouped by location
$C \leftarrow A \otimes B$	all pairs team-location are potential candidates†
$D \leftarrow C[O_1 \supseteq O_2][N,T,L]$	test if team operations include all operations at the location

* A modified version of this query will be seen in Q_5.

† Each block of C contains all concatenated pairs of tuples formed from one block of A and one block of B.

A

N	T	O
b	1	7
a	1	3
a	1	12
a	1	7
c	2	3
c	2	12
d	2	3
e	2	3

B

O	L
7	x
3	x
12	y

D

N	T	L
b	1	x
a	1	x
b	1	y
a	1	y
c	2	y
d	2	y
e	2	y

Table 3: Processing Q_1

Q_2: find the managers who earn more than all their employees.

For servicing this query the "." modifier is needed. When comparing two sets s_1 and s_2 by one of $\{\leq,<,\geq,>\}$ one may want to test the condition either for one or for all members of each set. For example:

some element of s_1 < all elements of s_2

which is equivalent to testing if (cf. [11]):

$\min(s_1) < \min(s_2)$

In the present notation the "." modifier is used on the side (sides) of the operator where the word all would appear. The several possibilities are shown in table 4.

s_1 \ s_2	some	all
some	$<,\leq,>,\geq$	$<.,\leq.,>.,\geq.$
all	$.<,.\leq,.>,.\geq$	$.<.,.\leq.,.>.,.\geq.$

Table 4: Quantified comparisons

operations	comments
$A \leftarrow \hat{S}[N,S]$	each employee as a possible manager
$B \leftarrow \overset{\vee}{S}[S,M]/M$	employees grouped by their managers
$C \leftarrow (A \otimes B)[N=M]$	correspondence manager-employees
$D \leftarrow C[S_1 > .S_2][N]$	condition about salaries

A

N	S
a	20
b	30
c	25
d	20
e	20

B

S	M
20	b
30	u
20	u
25	d
20	d

$\overset{\vee}{D}$

N
b

Table 5: Processing Q_2

Q_3: find the employees in "acceptable" teams, where a team is acceptable if all present employees (in $\overset{\vee}{S}$) earn at least as much as the prospective ones (in $\overset{\vee}{W}$).

operations	comments
$A \leftarrow \overset{\vee}{S}[N,S,T]/T$	old employees grouped by team
$B \leftarrow \overset{\vee}{W}[N,S,T]/T$	new employees grouped by team
$C \leftarrow (A \otimes B)[T_1 = T_2]$	correspondence:same team
$D \leftarrow C[S_1 .\geq .S_2][N_1,T_1] \oplus C[S_1 .\geq .S_2][N_2,T_2]$	merge teams satisfying the salary condition

A

N	S	T
a	20	1
b	30	1
c	25	2
d	20	2
e	20	2

B

N	S	T
f	23	1
g	15	1
h	20	2
i	18	2
j	15	2

$\vee$D

N	T
c	2
d	2
e	2
h	2
i	2
j	2

Table 6: Processing Q_3

Q_4: find the employees who earn more than all employees of lower category.

operations	comments
$A \leftarrow \hat{S}[N,C,S]$	each employee
$B \leftarrow \hat{S}[C,S] \oplus \{0,0\}$	categories and salaries, plus "dummy" category (see below)
$C \leftarrow (((A \otimes B)[C_1 > C_2])*I)/N$	correspondence: for each employee, all employees of lower category
$D \leftarrow C[S_1 >. S_2][N]$	salary condition

A

N	C	S
a	1	20
b	2	30
c	1	25
d	2	20
e	2	20

B

C	S
1	20
2	30
1	25
2	20
0	0

C

N	C_1	S_1	C_2	S_2
a	1	20	0	0
b	2	30	1	20
b	2	30	1	25
b	2	30	0	0
c	1	25	0	0
d	2	20	1	20
d	2	20	1	25
d	2	20	0	0
e	2	20	1	20
e	2	20	1	25
e	2	20	0	0

$\vee$D

N
a
b
c

Table 7: Processing Q_4

The "dummy" category was introduced in order to cause the acceptance of the employees of lowest category (for which there is no lower category to compare).

Q_5: find the pairs of teams and locations, such that the team has participated in all operations taking place at the location ; together with each team give also those of its employees who have actually worked at the location.

This is a variation of Q_1. One way to service the query is to perform the same operations executed for Q_1 and, in addition:

operations	comments
$E \leftarrow \breve{R}[N,T,L]$	employees effectively working at the locations
$F \leftarrow (\hat{D} \times \hat{E})[(N_1,T_1,L_1)=(N_2,T_2,L_2)]$	intersection with result of Q_1 *
$G \leftarrow F[N_1,T_1,L_1] * N_1$	result of Q_1 "filtered"

E

N	T	L
b	1	x
a	1	x
a	1	y
c	2	x
c	2	y
d	2	x
e	2	x

G

N	T	L
b	1	x
a	1	x
a	1	y
c	2	y

Table 8: Processing Q_5

* (N_1,T_1,L_1) etc. correspond to the domain-identifying lists of [5]

The proposed algebra is complete in the sense of [5] as shown below:

relational algebra	algebra of quotient relations
projection: $R[\underline{A}]$	$\check{R}[\underline{A}]$
restriction: $R[\underline{A}\ \theta\ \underline{B}]$	$\hat{R}[\underline{A}\ \theta\ \underline{B}] * I$
join: $R[\underline{A}\ \theta\ \underline{B}]S$	$(\hat{R} \times \hat{S})[\underline{A}\ \theta\ \underline{B}] * I_{1,2}$
division: $R[\underline{A} \div \underline{B}]S$	$((R/\bar{A} \otimes \check{S}[\underline{B}])[\underline{A} \supseteq \underline{B}])[\bar{A}]$
Cartesian product: $R \otimes S$	$\check{R} \otimes \check{S}$
union: $R \cup S$	$\check{R} \oplus \check{S}$
intersection: $R \cap S$	$((\hat{R} \otimes \hat{S})[I_1 = I_2])\ [I_1] * I_1$
difference: $R - S$	$((\hat{R} \otimes \check{S})[I_1 \not\approx I_2])\ [I_1] * I_1$

3. IMPLEMENTATION CONSIDERATIONS

The examples of the preceding section illustrate a process consisting of the three stages below; here, a sequence of operations for servicing a query will be regarded as a single"big operation".

a. The operands are partitioned. This stage can be executed in parallel, handling each operand separately.

b. A correspondence between blocks of the two operands is determined. In a block the values of the attributes by which the partitioning is made are obviously the same in all tuples of the block; since the correspondence between blocks depends only of such "representative" values, this stage can be performed very efficiently.

c. For the blocks that do correspond some condition is then investigated. Even here an improved performance may be obtainable: not all tuples in a block need be examined in certain cases (e.g. when "$\approx$" is used to compare sets we may "accept" the block as soon as the first match occurs). Also, for order comparisons, the minimum or maximum values of sets can be determined separately, as in stage a, for each operand; then only one comparison will be needed to test the condition.

The three stages are easily identified when a query is expressed in relational calculus. Consider, for example, Q_3 (ignoring the final merge, for simplicity):

$$(s,w):\ \forall s'\ \forall w'\ \underbrace{(s'[T] = s[T] \wedge w'[T] = w[T]}_{\text{partitioning}}$$

$$\wedge \underbrace{s[T] = w[T]}_{\text{correspondence between blocks}})$$

$$\rightarrow \underbrace{s'[S] \geq w'[S]}_{\text{condition}}$$

Certain efficiency considerations often made with respect to the relational algebra are also applicable here. Clearly, one should avoid the actual computation of a Cartesian product; by looking ahead to the next (simple) operations (especially projections) it may be seen if the desired result will be in fact a concatenation of tuples, or a merge, or if one of the operands is simply used for restricting the other one (see Q_1 , Q_3 and Q_2, respectively, and cf. the "keep" feature in [12])*.

Several ways to implement partitioning can be suggested.Two will be mentioned here:

- pointer arrays;
- aggregate inverted lists.

A pointer array P indicating the first tuple in each block of a quotient relation can be used if the tuples are previously rearranged.†

Table 9 reproduces the example in table 2, where $Z \leftarrow \overset{\vee}{R}/\{T,L\}$ is shown.

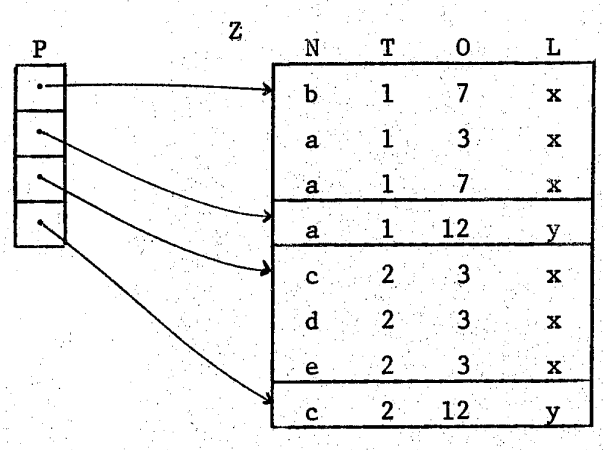

Table 9: Partitioning implemented by pointer array

* Another strategy is to introduce new notation, as for example R ⊗ [<condition>] S (for join), R ⊕ [<condition>] S (for conditional union), R ⊖ [<condition>] S (for binary restriction). We have refrained from introducing new notation (except when indispensable, for the sake of readability.

† Sorting on the attributes by which the relation is partitioned is one obvious way to accomplish this rearrangement.

Alternatively, P could point to another pointer array Q, which in turn would show the quotient relation rearranged. This level of indirection would be used if physical rearrangement of the tuples is not desired.

Aggregate inverted lists for a set of attributes A are obtainable, for example, by intersecting the inverted lists for the individual attributes in A (if such inverted lists are available).

A sort routine may be a major tool in an implementation. Sorting on the atributes in A (in any sequence) is one way to perform the partitioning; sorting also on "pivot" attributes may be useful, leading to algorithms linear in the number of comparisons [16]; sorting on attributes in A in a specified sequence can provide a hierarchical presentation of the data for output.

Partitioning, on the other hand, may be considered as a factor for deciding how to organize secondary storage. If the same partitioning scheme is used very often, it may be useful to keep the respective quotient relation, establishing a correspondence between its blocks and physical pages.

4. EXTENSIONS

Practical usage often requires more than a minimal set of operators. Thus, we would not recommend that only the six operators presented in section 2 be employed; other operations may be introduced whenever convenient, their definition being given in terms of the basic ones.

In particular, join, intersection and difference are obvious choices. Other posssible candidates will be mentioned in the sequel, but no additional notation will be proposed.

- Conditional expressions involving more than one comparison and including logical connectives, to be used with restriction and join, have been suggested [13,14].

- In query Q_4 in section 2 we had a situation where a block (representing one employee, of a certain category) had to be compared with a block consisting of the employees of lower category. Conceptually, this second block is obtained, for a given category c, by separating $\breve{R}$ into two groups of tuples, depending on their involving or not a category lower than c; the first group constitutes the desired block.

Notice that this separation must be done for each value c. The operation involved is completely different from partitioning $\breve{R}$ by category; in fact, here we are dealing with a mathematical relation ("<") which is not an equivalence relation. In [15] page 108 the operation, with the name of split, is introduced as a basic operation on ordered sets.

- Transitive closure is another useful operation. Take a relation $V \subseteq N \times M$, where N is employee name and M is manager. If we want to find the pairs consisting of an employee and the manager of his manager we can write in the present algebra:

$$A \leftarrow ((\breve{V} \otimes \breve{V})\ [M_1 = N_2])\ [N_1, M_2]$$

which may be merged with $\breve{V}$ to give, for each employee, two levels of managers

to whom he reports:

$$B \leftarrow \overset{\vee}{\nabla} \oplus A$$

and this process may be repeated to m levels of managers. A further generalization would leave m unbounded - additional levels would be added until the result for k+1 levels contains the same pairs as for k levels.

- We may want to reduce a quotient relation, through its decomposition into quotient relations of lesser degree. Consider $Y \leftarrow \overset{\vee}{R}[R,L,O]$, with $\overset{\vee}{R}$ as given in section 2; the computations*:

operations	comments
$A \leftarrow \overset{\vee}{Y}[L,O] / L$	projection
$B \leftarrow ((\overset{\vee}{Y}/\{T,L\} \otimes \overset{\vee}{A}/L)[(L_1,O_1) \supseteq (L_2,O_2)][T,L_1]$	similar to divison, with second operand partitioned by non-empty set of attributes.
$C \leftarrow (\overset{\vee}{Y} /\{T,L\} \otimes \overset{\vee}{B})[(T_1,L_1) \% (T_2,L_2)]$	difference with respect to a specified set of attributes

yield the results in table 10.

Y

T	L	O
1	x	7
1	x	3
1	y	12
2	x	3
2	y	12

A

L	O
x	7
x	3
y	12

B

T	L
1	x
1	y
2	y

C

T	L	O
2	x	3

Table 10: Example of reduction

* Reduction resembles the familiar arithmetic process of finding "quotient" (B) and "remainder"(C), given a "dividend"(Y) and a "divisor"(A).

and computing $(((B \otimes A)\ [L_1 = L_2])\ [\bar{L}_2] \oplus C) * I$ will reconstruct Y. The formulas given above can be generalized to:

$$D \leftarrow \overset{\vee}{Z}[A]/C$$
$$Q \leftarrow ((\overset{\vee}{Z}/B \otimes \overset{\vee}{D}/C)[A_1 \supseteq A_2]\ [B]$$
$$R \leftarrow (\overset{\vee}{Z}/B \otimes \overset{\vee}{Q})[B_1 \not\approx B_2]$$
$$\overset{\vee}{Z} \leftarrow (((Q \otimes D)[C_1 = C_2]\ [\bar{C}_2] \oplus R) * I$$

with the requirement that $A \cup B = I$. In the example:

$$A = \{L,O\}$$
$$B = \{T,L\}$$
$$C = A \cap B = \{L\}$$

We say that a quotient relation is <u>reducible</u> if, for some choice of A and B, $R = \phi$. Whenever $C = \phi$ the formulas for D,Q and for the recomposition of $\overset{\vee}{Z}$ become:

$$D \leftarrow \overset{\vee}{Z}[A]$$
$$Q \leftarrow ((\overset{\vee}{Z}/B \otimes \overset{\vee}{D})[A \supseteq A])[B]$$
$$\overset{\vee}{Z} \leftarrow ((Q \otimes D) \oplus R) * I$$

so that the formula for Q becomes equivalent (cf. end of section 2) to the division of relational algebra.

5. CONCLUSIONS AND FUTURE WORK

The proposed algebra of quotient relations incorporates the well-known algebraic notion of partitioning, which already appears as a language feature in relational languages [8,9]*.

It provides a more direct way to express certain queries than what is offered by the original relational algebra. In the latter it is easy to express the "$\wedge$", "$\vee$", "$\sim$" logical connectives (through intersection, union and difference); however, "$\rightarrow$" has no direct counterpart ($p \rightarrow q$ must be first changed into $\sim p \vee q$, which leads to "complemented" queries). As an example, query Q_1 in section 2 would be in relational algebra:

operations	comments
$A \leftarrow R[N,T] \otimes R[O,L]$	all possible combinations
$B \leftarrow A[T,O,L] - R[T,O,L]$	missing triples
$C \leftarrow R[T,L] - B[T,L]$	pairs which do not correspond to any missing triple
$D \leftarrow (R[N,T][T=T]C)[N,T,L]$	append employees to accepted pairs

Also, as argued in section 3, certain advantages in terms of efficiency may be gained. These will be the object of further investigation, especially in the context of sequences of operations.

The authors feel that an algebraic language is an adequate vehicle for expressing formally the semantics of relational languages. An algebraic language can also be an intermediate language; procedural in that it indicates how queries are to be serviced, and yet independent of the chosen data structures; translators from high-level relational languages into the present algebra will be the other main subject of investigation.

---

* cf. also the "glumps" in [1].

ACKNOWLEDGEMENTS

The authors are grateful to M. Stonebraker, E. Neuhold and C. Date for several helpful discussions. Any errors, are of course the authors' sole responsibility.

REFERENCES

1. CODASYL - "An Information Algebra" - CACM 5(1962) 190-204.

2. Kobayashi, I - "Information and Information Processing Structure" - Information Systems 1(1975) 39-49.

3. Childs, D.L. - "Feasibility of a Set Theoretic Data Structure" - IFIP (1968).

4. Codd, E.F. - "A Relational Model of Data for Large Shared Data Banks" - CACM 13 (1970) 377-387.

5. Codd, E.F. - "Relational Completeness of Data Base Sublanguages" - in Data Base Systems (1971).

6. Kerschberg, L., Klug, A. and Tsichritzis, D. - "A Taxonomy of Data Models" 2nd VLDB Conference (1976).

7. ANSI - Interim Report ANSI/X5/SPARC Study Group on Data Base Management Systems (1975).

8. Chamberlin, D. and Boyce, R -"SEQUEL: A Structured English Query Language" ACM SIGMOD (1974).

9. Stonebraker, M.R. and Wong, E. - "INGRES: A Relational Data Base System"- National Computer Conference (1975).

10. MacLane, S. and Birckhoff, G. - "Algebra" - Macmillan (1968).

11. Boyce, R.F. et al - "Specifying Queries as Relational Expressions" - CACM 18 (1975) 621-628.

12. Tsichritzis, D. - "LSL - A Link and Selection Language" - Univ. Toronto T.R. CSRG - 61 (1975).

13. Pecherer, R.M. - "Efficient Evaluation of Expressions in a Relational Algebra" - Univ. of California Berkeley T.R. ERL - M510 (1975).

14. Smith, J.M.S. - "Optimizing the Performance of a Relational Database Interface" - CACM 18(1975) 568-579.

15. Aho, A., Hopcroft, J. and Ullman, J. - "The Design and Analysis of Computer Algorithms" - Addison-Wesley (1974).

16. Furtado, A.L. and Brodie, M.L. - "A Data Structure for Fast Relational Algebra Operations" - PUC/RJ - T.R. 7/76 (1976).