PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 7/76

A DATA STRUCTURE FOR FAST RELATIONAL ALGEBRA OPERATIONS

by

Antonio Luz Furtado

and

Michael L. Brodie

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 7/76

A DATA STRUCTURE FOR FAST RELATIONAL ALGEBRA OPERATIONS*

by

Antonio Luz Furtado

and

Michael L. Brodie

Series:Editor: Sérgio E. R. Carvalho

April, 1976

* This work was supported in part by the Brazilian government Agency FINEP, under contract nº 244/CT and developed by the Data Base group of the Depto. de Informática - PUC-R.J.

** Currently visiting from the Department of Computer Science of the University of Toronto, Canada, under the sponsorship of the Canadian International Development Agency.

Copies may be requested from:

Rosane T.L. Castilho, Head
Setor de Documentação e Informação
Deptº de Infromática - PUC - RJ
R. Marquês de São Vicente, 209 - Gávea
20000 - Rio de Janeiro - RJ - BRASIL

ACKNOWLEDGEMENT

The authors are indebted to G.C. Magalhães, N. Ziviane T.H.C. Pequeno, participating in the Hyades project.

ABSTRACT:

A logical design for the implementation of fast relational algebra operations, in the context of a query - oriented data base, is described. The logical data structure and the algorithms are presented and evaluated.

KEY WORDS:

Relational algebra, data structures, data bases.

RESUMO:

Um projeto lógico para a implementação de operações de algebra relacional rápidas, no contexto de um banco de dados orientado para consulta, é descrito. A estrutura de dados lógica e os algoritmos são apresentados e avaliados.

PALAVRAS CHAVE:

Algebra relacional, estruturas de dados, bancos de dados.

CONTENTS

1. INTRODUCTION

The major feature of the relational model of data [1] appears to be the data independence achieved by the high logical level presented to the user. At the same time, the existence of an efficient implementation for large shared data bases has been questioned, due in part to this absence of physical characteristics on which to base such an implementation. In this paper, we present a logical design for the data structures and the corresponding algorithms for the relational algebra operations. The evaluation of the algorithms strongly indicates the possibility of an efficient implementation.

This paper is part of the HYADES project at the Pontificia Universidade Católica do Rio de Janeiro (PUC-RJ) in which we are implementing a prototype query-oriented relational data base management system. The prototype will be made available to other Brazilian Universities and will be used as a basis for further research. The subject was described formally [2], and this theoretical framework has determined the following five design principles. First, the relational model of data will be used. We will assume that the reader is familiar with the concepts and terminology of the basic relational model [1,3]. Second, in an attempt to achieve a unified and complete data base description technique, the concept of a directed hypergraph over domains was introduced [4]. Third (following from the second principle), domains are to be treated equally with relations as basic components of data base description and operation. Fourth (a consequence of two and three above), all relations, both primary and those derived from primary relations [5], are treated identically. Finally, as

many relational algebra operations as possible are to be performed on fast access inverted lists for attributes rather than on the relations themselves.

2. THE DATA STRUCTURE

This section describes the four constituent structures which compose the logical data structure. They are; the CLASS, the DOMAIN, the INVERTED LIST for anattribute and the RELATION.

A class is the union of all domains which are declared to be θ-comparable,i.e.,they may be the pivot domains in θ-joins or θ-restrictions. The CLASS structure is an ordered list of values. It is a data pool in that actual data values appear only once in the data base, in these structures. The structure allows only read, insertion and deletion which are done using an order preserving storage mapping function [6]. In the above ways underlying domains provide θ-comparability, non-redundancy and a degree of consistency.

A domain is represented in the DOMAIN structure which has a two pointer entry for each existing value. One pointer is for the actual value in the CLASS; the second is for the inverted list for the domain entry. As with CLASSes the entries are ordered on value (index to the CLASS) and may be inserted or deleted via an order preserving storage mapping function, for efficient access. Here again, θ-comparability and a degree of non-redundancy are provided.

POINTER TO THE VALUE OF X IN THE CLASS STRUCTURE	POINTER TO FIRST OCCURRENCE LIST FOR X

Fig.1.1. DOMAIN ENTRY TEMPLATE FOR VALUE X

Let us consider an attribute, A, to be the role a domain plays in one relation, R. The INVERTED LIST of A in R, denoted $\hat{R}[A]$ (in this paper hat, $\hat{}$, denotes on ordered list) is represented by a set of OCCURRENCE LIST structures. For each entry in the DOMAIN there is an OCCURRENCE LIST . In fact, for each attribute based on the DOMAIN there is an OCCURRENCE LIST for each entry of the DOMAIN. All OCCURRENCE LISTs based on a given DOMAIN entry are linked from the DOMAIN entry through their headers.

ROLE NAME	POINTER TO FIRST OCCURRENCE CELL	POINTER TO NEXT OCCURRENCE LIST HEADER

Fig.1.2. OCCURRENCE LIST HEADER TEMPLATE

For one value, v, of a given attribute, A, the OCCURRENCE LIST , denoted $\hat{R}[A=v]$, is a set of tuple identifiers for tuples containing that attribute value. An OCCURRENCE LIST is maintained as a linked queue of OCCURRENCE CELLs. Hence, the queue is ordered on tuple identifier since tuples may be inserted only at the end of a relation.

POINTER TO NEXT OCCURRENCE CELL	TUPLE IDENTIFIER

Fig.1.3. OCCURRENCE CELL TEMPLATE

A relation of degree n with m tuples is represented by RELATION, a rectangular table of m rows and n columns. Each table element has two pointers: one references the actual value in the CLASS, the second points to its OCCURRENCE CELL representation. Each tuple has a row index which is the tuple identifier. Tuples may be inserted only at the bottom of the table. This structure is accessed as little as possible by the algorithms.

POINTER TO THE VALUE IN THE CLASS STRUCTURE	POINTER TO THIS ENTRY'S OCCURRENCE CELL

Fig.1.4. RELATION ENTRY TEMPLATE

Figure 1.5 uses the four data structures to represent (partially) a relation R with attribute A based on domain D and class,C.

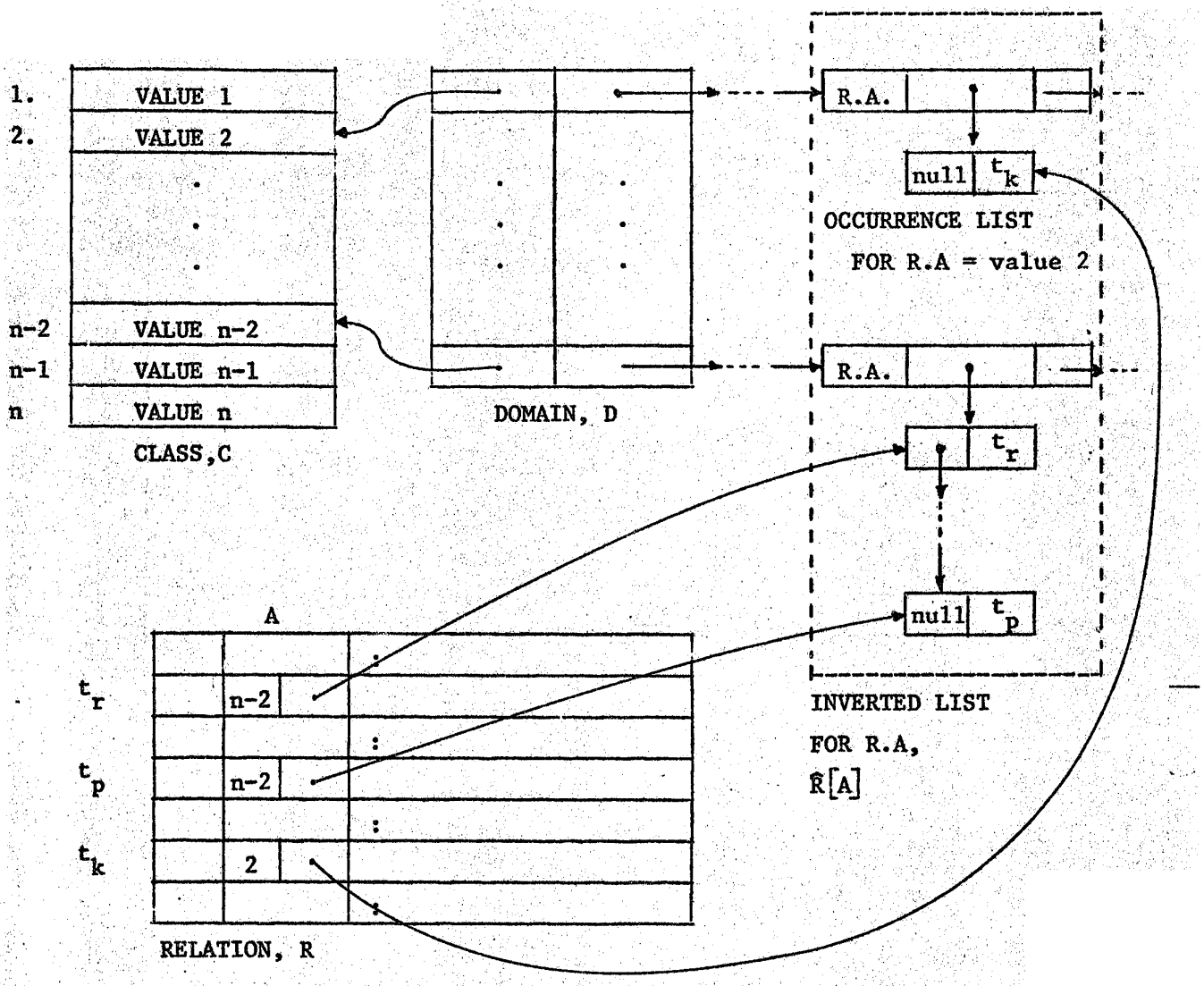

Fig. 1.5. (PARTIAL) REPRESENTATION OF R(...,A,...) WITH A BASED ON DOMAIN D AND CLASS C.

Note: t_k contains the only occurrence of A=2
t_r contains the first occurrence of A=n-2
t_p contains the last occurrence of A=n-2

The prime characteristic of the above data structures is that they are all ordered on their operational characteristic: classes on actual value, domains on value index and occurrence lists on tuple identifier. Further, due to the order preserving storage mapping function used for CLASS, class indices in DOMAIN and RELATION may be treated as actual data values for purposes of comparison. Aspects such as physical storage details and update characteristics were not described as they are not within the scope of this paper. For simplicity, we have also omitted optimization details, such as extra pointers and data base statistics, and the description of the data dictionary or directory function.

3. RELATIONAL ALGEBRA OPERATIONS

Here, the relational algebra operations [3] are considered to be a sequence of set theoretic operations on the inverted lists for attributes for the most part but in some cases on the relation itself. These operations are particulary fast since not only is the inverted list ordered on attribute value but also the occurrence list is on ordered set of tuple identifiers. Hence, each relation can be accessed in an order based an any one attribute. Relational operations based on simple domains require no sorting; however, non-simple domains require further sorting for the sake of efficiency.

In the sequel, a general sort algorithm is presented. Then, using the sort, an algorithm for each relational algebra operation will be presented and analyzed for order. The result, T, of a relational algebra operation can be seen as a list of tuple identifiers. For the operations set difference,

intersection, Θ-restriction, projection and division, T is a list of single tuple identifiers refering to the relation which is the first operand in the relational algebra expression. For Θ-join, and extended cartesian product T is a list of tuple identifier pairs, one for each relation in the relational algebra expression. For union, T is one list of tuple identifiers for each of the two relations concerned.

3.1. SORT R on attributes A

This algorithm sorts a relation, R, on the ordered set of attributes $A = \langle A_1, A_2 \ldots, A_k \rangle$ where the major to minor order is left to right. The desired result will be denote $\hat{R}\langle A_1, A_2, \ldots, A_k \rangle$. The algorithm is a radix sort which starts at A_k and steps through A using a minor order attribute, A_{i-1}, to sort the next attribute A_i. A step is conducted by reordering the occurrence cells, for duplicate attribute values for A_i, in each occurrence list depending on the lower order attribute values. Hence, $\hat{R}[A_{i-1}]$ is reordered based on the reordered $\hat{R}[A_i]$: the result is $\hat{R}'[A_{i-1}]$ which contains the tuple identifiers ordered for $\hat{R}\langle A_{i-1}, A_i, \ldots, A_k \rangle$. The algorithm is complete when $\hat{R}'[A]$ contains the desired result for T. Let us assume $\hat{R}'[A_i]$ is initially empty for all i.

SORT Algorithm

1. For each attribute A_i in $\langle A_1, A_2, \ldots, A_k \rangle$ i = k to 2
2. For each attribute value v in $\hat{R}[A_i]$
3. For each tuple occurrence, t, containing v in $R'[A_i = v]$
4. Get the corresponding A_{i-1} value, $x = r_t[A_{i-1}]$
5. Append t to $R'[A_{i-1} = x]$
6. $\hat{R}[A_{i-1}] \leftarrow \hat{R}'[A_{i-1}]$

Order

Step 1 is executed at most k-1 times since $\hat{R}[A_k]$ contains an acceptable ordering for $\hat{R}<A_k>$ If A contains attributes which are candidate keys (simple or composite) then the sort may start at the leftmost such attribute. Since $R[A_i]$ exists as as DOMAIN and $\hat{R}[A_i]$ exists as an inverted list steps 2 and 3 are linear in the cardinality of the respective sets. Step 4 accesses the RELATION via the tuple identifier in the occurrence cell and retrieves x directly. $\hat{R}'[A_{i-1}]$ is constructed using a queue for each value in $R[A_{i-1}]$. In fact, $\hat{R}'[A_{i-1}]$ is not assigned to $\hat{R}[A_{i-1}]$; $\hat{R}'[A_{i-1}]$ is used directly in the next step. Hence, the order is

$$\sum_{i=k}^{2} \sum_{v \in A_i} |\hat{R}[A_i = v]| = (k-1) \times t \text{ where } t = \sum_{v \in A_i} |\hat{R}[A_i = v]| =$$

$|\hat{R}[A_i]| = |R|$ that is, the maximum order of the sort is the number of tuples times one less than the number of attributes in the attributes in the attribute ordering, $|R|\ (n_A - 1)$[1].

The sort algorithm could also be performed <u>in situ</u> on the occurrence lists. To do this the pointer to entry's occurrence cell, in RELATION, is used (see Figs. 1.4 and 1.5). This strategy has the advantage of providing, without additional space requirements, the "partial sorts" $\hat{R}<A_j, A_{j+1}, \ldots, A_k>$. Knowledge of the availability of such sorts can be useful if the optimization of sequences of relational algebra operations is to be attempted [7]. However, a major disadvantage is that the order

---

[1] Note: n_x denotes the number of domains in the domain list x. In the above sort $n_A = k$.

by tuple identifiers within the occurrence lists is destroyed.

An interesting property of the sort algorithm is observed when sorting R on two domains, say <A,B>. Suppose that when "navigating" from B to A through R we form lists of the form $L[A=a]$ which corresponds to the re-ordered occurrence list $R[A=a]$. Here, each tuple identifier is replaced by the $b \in B$ appearing in the same tuple of R as $a \in A$. In these lists, duplicates from B are easily eliminated due to the sorting.

The lists allow us to answer the queries:

a. what elements from A correspond through R to <u>some</u> element from B?

b. what elements from A correspond through R to <u>all</u> elements from B?

The answers are obtained, respectively, by accepting the elements from B satisfying:

i. $|L[A=a]| \geq 1$ i.e., one element of the projection

ii. $|L[A=a]| = |R[B]|$ i.e., one element of the division

The same queries can be answered if we use the inverse sequence B,A , and take, respectively, the results of:

i'. $\bigcup_{b \in R[B]} L[B=b]$ i.e., the complete projection

ii'. $\bigcap_{b \in R[B]} L[B=b]$ i.e., the complete division

Let us consider this "navigation" in on intuitive way.

The general problem of implementing the relational algebra operations is to translate a logical operation on relations and attributes into physical access paths from attribute to attribute. This concept is emphasized by describing a data base using directed hypergraphs and is similar those ideas presented in SQUARE [10] and the Link and Selector Language [9].Due to the logical data structure, presented here, all algorithms operate as though they were "navigating" from attribute to attribute. It will be seen that in the case of simple domains the navigation is direct , from one attribute to another using inverted lists. In general, navigation is from one attribute through a relation to another attribute in the same relation or to a second relation and then to an attribute in that final relation. On this "navigation" process the relational algebra operations collect data in order to answer the query and in order to navigate further (pivotal information).

3.2. Set Theoretic Operations $R \cup S$, $R \cap S$, $R - S$, $R \oplus S$

Union, intersection and set difference of two degree r , θ-comparable relations R and S are particularly simple. First, R and S are sorted on a common attribute order. Then, a merge is conducted using 'and' for inter - section, 'or' for union and 'and not' for set difference. The result for intersection and set difference is a set of tuple indentifiers for R. Union results in two lists of tuple identifier pairs; one for R and one for S.

Set Theoretic Algorithm

1. Sort R on A, where A is the common attribute order
2. Sort S on A
3. Merge R and S using and ($\cap$), or ($\cup$) or and not(-)

Order

Steps 1 and 2 use the above sort. Step 3 can be performed by comparing, in order, the tuples of R with those of S, e.g., n comparisons per tuple. Hence, the maximum order is the sum of the step orders or $O((n_A - 1)(|R| + |S|) + n_A(|R| + |S|))$ which is roughly the order of the sort.

The relational algebra includes the extended cartesian product on sets S_i, denoted $S_1 \otimes S_2 \otimes \ldots \otimes S_m$, more for completeness than for practical use. The standard algorithm is of order $\prod_{i=1}^{m} |S_i|$, with respect to tuple catenation; however, no comparisons are involved.

3.3. Projection, R[A]

A projection is a partitioning of the tuples of a relation, R, into groups in which the domains of interest, A, have equal values. If A is simple the projection is achieved by traversing the inverted list $\hat{R}[A]$ selecting one (say the first) occurrence cell from each occurrence list, hence an order $|\hat{R}[A]|$ operation. In general, A is non-simple and R must be sorted on A for an efficient projection. The projection can be constructed during the sort by maintaining pointers within each occurrence list to mark those tuples for which all preceeding domain values were

equal. Pointers would be handled in step 5 of the sort algorithm; hence affect the order by a constant term. Otherwise, we could step through the relation using the ordering in $\hat{R}[A]$ making n_A comparisons at each step. In either case, the order of the algorithm is $O(n_A \times |R|)$. Again, the order is dominated by the sort.

3.4. JOIN $R[A \theta B]S$ $\theta \in \{<, \leq, =, \neq, \geq, >\}$

For the θ-join of relations R and S on the θ-comparable domains A and B this algorithm provides a list of tuple identifier pairs (r,s) such that the corresponding tuple in the resultant relation is formed from the catenation $R_r \cap S_s$. If A and B are simple domains the algorithm is performed on the inverted lists $\hat{R}[A]$ and $\hat{S}[B]$ directly. For A and B non-simple domains the algorithm is performed on the relations R and S. Let us consider the two cases separately.

θ-JOIN Algorithm For A and B Simple Domains

1. For each attribute value a in $R[A]$
2. For each attribute value b in $S[B]$ with $a\theta b$
3. For each tuple t in $\hat{R}[A=a]$
4. For each tuple k in $\hat{S}[B=b]$
5. Add the tuple pair (t,k) to the result.

Order

For each value a in step 1, step 2 divides the $\hat{S}[B]$ into those elements that are θ-related to a and those that are not. Since both $\hat{R}[A]$ and $\hat{S}[B]$ are ordered lists, steps 1 and 2 are effected by a merge in

$O(|R[A]| + |S[B]|)$ operations. This is obvious for $\Theta \in \{=,\neq\}$, but let us consider $\Theta \in \{<, \leq\}$. Using these operators for a given value a in $\hat{R}[A]$ we find the first b in $\hat{S}[B]$ for which $a\Theta b$ is true; b and all its sucessors in $\hat{S}[B]$ are Θ-related to a. Note that for $a_1<a_2$ with $a_1\Theta b$ false, $a_2\Theta b$ is also false. Hence, the partition points are found using a merge. For $\Theta \in \{>, \geq\}$ the above method applies except that $\hat{R}[A]$ and $\hat{S}|B|$ are traversed from bottom to top. For each pair (a,b) found in steps 1 and 2, steps 3,4 and 5 take the cartesian product of the occurrence lists in $O(|R[A=a]| \times |S[B=b]|)$. Hence, the order of the Θ-join algorithm for simple domains is:

$$O(|R[A]|+|S[B]|) \text{ comparisons} + O\left(\sum_{a\in R[A]} \sum_{\substack{b\in S[B] \\ a\Theta b}} |\hat{R}[A=a]| \times |\hat{S}[B=b]|\right) \text{catenations.}$$

<u>Θ-JOIN Algorithm For A and B non-simple Domains</u>

1. SORT R on A
2. SORT S on B
3. For each tuple t in $\hat{R}$
4. For each tuple k in $\hat{S}$
5. if $r_t[A]\ \Theta\ s_k[B]$ then add(t,k) to the result.

<u>Order</u>

The sort algorithm produces two ordered list $\hat{R}$ and $\hat{S}$. Like the simple domain counterpart, a merge is used to find all (a,b) pairs in steps 3 and 4; however, each step requires n_A comparisons rather than one. When a pair is found only one catenation occurs rather than many as in the simple domain algorithm. The order of the Θ-join algorithm on non-simple

domains is the sum of the orders of the sort, the comparisons in the merge and catenations for (a,b) pairs:

$$O((n_A - 1)x(|R| + |S|)) + O(n_A x(|R| + |S|)) \text{ comparisons}$$

$$+ O(\sum_{a \in R[A]} \sum_{\substack{b \in S[B] \\ a \Theta b}} 1) \quad \text{catenations.}$$

For both join algorithms let us consider the number of catenations by estimating the number of summands in the corresponding double sum expressions. The simplest case the equi-join, or = -join, on key domains A and B . For simple domains occurrence lists contain one element, $|R[A=a]| = 1$ for all a in $\hat{R}[A]$ and similarly for $\hat{S}[B]$. Hence, the number of catenations is the number of summands which, in turn, is at most $|R[A]|$. In this case the order of the equi-join algorithm is $O(|R[A]| + |S[B]|)$ or linear in the number of tuples. Similarly for non-simple domains the number of catenations is at most $O(|R|)$ which produces a total order of $O(n_A(|R| + |S|))$, again sort dominated. For A and B non prime the double sum depends on the operation Θ , the distribution and the frequency of occurrence of values in $R[A]$ and $S[B]$.

3.5. Θ-Restriction $R[A\Theta B]$

There is a close analogy between this operation and the Θ-join. As with the Θ-join we will consider two algorithms; the first operates on inverted lists and may be used for simple domains with many duplicate values, the second accesses the relation and is used for non-simple domains or for simple domains with few duplicate values.

Θ-Restriction Algorithm 1

1. For each value a in $\hat{R}[A]$
2. For each value b in $\hat{R}[B]$ with $a\Theta b$
2. Add $\hat{R}[A=a] \cap \hat{R}[B=b]$ to the result

Order

Steps 1 and 2 are identical to the Θ-join. Since the occurrence lists in step 3 are ordered, the operation is merge-efficient in comparisons. The final order is:

$$O\left(|R[A]| + |S[B]| + \sum_{a \in R[A]} \sum_{\substack{b \in R[B] \\ a\Theta b}} \left(|\hat{R}[A=a]| + |\hat{R}[B=b]| \right)\right) \text{ comparisons}$$

In the worst case this becomes $O(|R[A]| + |S[B]| + 2|R|) \leq O(|R|)$

Θ-Restriction Algorithm 2

1. For each tuple t in R
2. if $r_t|A| \ \Theta \ r_t|B|$ then tuple t is in the Θ-restriction.

Order

The order of this algorithm is $(n_A \times |R|)$ since n_A comparisons are required for each tuple of R. Notice that, unlike the Θ-join algorithm, no sorting is required.

3.6. Division $R[A \div B]S$

The division $R[A \div B]S$ is the set of all tuples $r[\bar{A}]$[1] to which there corresponds (at least) all elements of $S[B]$. For example, the expression SUPLIER[PART-# ÷ PART-#]PART-USE refers to those suppliers who supply all parts. This algorithm produces a list of tuple identifiers for tuples in R.

Division Algorithm

1. PROJECT S on B
2. SORT R on $<\bar{A},A>$
3. partition R into the image sets $g_R(R[\bar{A}])$
4. For each image set $g \in g_R(R[\bar{A}])$ while $|g| \geq |S[B]|$
5. possible-member ← 'true'
6. For each value b in $S|B|$ while possible-member
7. if $b \notin g$ then possible-member ← 'false'
8. if possible-member then accept $r[A]$ corresponding to g.

Order

Steps 1 and 2 use previously defined algorithms. Note that PROJECT ensures that S is sorted on B. Step 2 in sorting R on $\bar{A}$ effects step 3 since the image sets $g_R(R[\bar{A}])$ are precisely the occurrence lists in $\hat{R}[\bar{A}]$; further individual image sets are ordered since we also have $\hat{R}[A]$. The final steps are performed on the sorted RELATIONs in a merge-like fashion. Note that each tuple of R appears in exactly one image set g. Also,

[1] Note: $\bar{A}$ is complement set of domains for A.

when comparing b to values in g, as long as the comparisons fail the next tuple of g is taken. If ever the comparison succeeds for some tuple y in g, the comparisons for the next value of $\hat{S}[B]$ will start with the tuple following y .Hence, each tuple of R is examined at most once. The order for these final steps is $O(n_A \times |R|)$. The order for the division algorithm is:

$$O(n_B \times |S| + (\bar{n}_{A,A}-1) \times |R| + n_A \times |R|) = O(n_B \times |S| + \text{degree}(R) \times |R|))$$

4. CONCLUSION

A data structure for a relational data base and the corresponding algorithms for the relational algebra operations have been presented. Relations are stored as relation tables and a collections of inverted lists for attributes. In some cases, algorithms access the relations directly. However,in most cases the inverted lists are accessed and the relation table may provide a means of navigating from are attribute to another. In the Hyades project all attributes possess inverted lists. It has been noted that for large data bases this is not advisable [8],but we can still claim that the algorithms depending on their availability described in this paper are applicable for operations on those having such lists. In this way, an efficiency close to the information theoretic lower bound (approximately linear in the number of tuples) is achieved. These orders, summarized in Table 4, strongly indicate the possibility of an efficient implementation of a relational data base management system.

Relational Algebra Operation	Order: Compares	Order: Catenations	Comment
SORT R on A	$\|R\| \times \eta_A$	$\|R\|$	
R S, R S, R-S	$\|R\| \times$ degree(R) $+ \|S\| \times$ degree(S)		sort dominated
R ⊗ S		$n_A \times n_B$	
R[A] (i) A simple	$\|R[A]\|$		
(ii) A non-simple	$n_A \times \|R\|$		sort dominated
R[AΘB]S			
(i) A simple	$\|R[A]\| + \|S[B]\|$	$\sum_{a \in R[A]} \sum_{b \in S[B]} (\|R[A=a]\| \times \|S[B=b]\|)$	
(ii) A non-simple	$n_A(\|R\| + \|S\|)$	$\sum_{a \in R[A]} \sum_{\substack{b \in B[B] \\ a \Theta b}} 1$	sort dominated comparisons
R[AΘB]			
(i) A simple	$\|R[A]\| + \|S[B]\| +$	$\sum_{a \in R[A]} \sum_{\substack{b \in R[B] \\ a \Theta b}} (\|R[A=a]\| + \|S[B=b]\|)$	
(ii) A non-simple	$n_A \times \|R\|$		sort dominated
R[A÷B]S	$n_B \times \|S\| + \|R\| \times$ degree(R)		sort dominated

Table 4 - Algorithm Orders For the Relational Algebra Operations

The HYADES project is currently investigating the implementation of the algorithms and data structures described above [11] . Further work in the project involves; the extension of the system to handle update, data definition and data manipulation language designs; a restricted natural language query front end and continued work on a semantic and descriptive model for relational data bases.

REFERENCES:

1 . CODD,E.F. A relational model for large shared data banks. CACM 13 (6): 377-87, june 1970.

2 . FURTADO, A. L. Formal aspects of the relational model. Rio de Janeiro, PUC, Depto. de Informática (to appear).

3 . CODD,E.F. Relational completeness of data base sublanguages. San Jose, IBM Research Laboratory, 1972 RJ-987.

4 . BRODIE,M.L., ed. ZETA: a prototype relational data base management system. Toronto, Univ. of Toronto, 1975. R.R.CSRG-51.

5 . BERGE,C. Graphs et hypergraphes. Paris, Dunod, 1970.

6 . STONEBRAKER,M.R. & WONG E. INGRES: a relational data base system. In: NATIONAL COMPUTER CONFERENCE, Anaheim, 1975

7 . SMITH,J.M.S. Optimizing the performance of a relational algebra database interface. CACM, 18 (10): S68-79, Oct.1975

8 . BACHMAN,C. The programmer as mavigator. CACM, 16 (11): 653 - 58, Nov. 1973.

9 . TSICHRITZIS,D. LSL: a link and selector language. Toronto, Univ. of Toronto, 1975. T.R. CSRG-61.

10. BOYCE,R.F. et alii: Specifying queries as relational expressions: SQUARE. In: ACM SIGPLAN-SIGIR INTERFACE MEETING, Gaithersburg, Ma., 1973 p.31-47.

11. MAGALHÃES,G.C. et alii., Especificação de uma interface para um banco de dados relacional. In: INTERNATIONAL SYMPOSIUM OF METHODOLOGIES FOR THE DESIGN AND CONSTRUCTION OF SOFTWARE AND HARDWARE SYSTEMS, Rio de Janeiro, 1976.