A MUMPS-BASED RELATIONAL DATA BASE SYSTEM (MRDB)

Georges S. Nicolas and John W. Lewis

Department of Electrical Engineering and
Division of Laboratory Medicine
Washington University, St. Louis, Missouri 63130

A relational database management system has been implemented in the MUMPS language using a binary tree structure for storage of relations. Several examples are given to illustrate the available functions and commands. Some details of the data structure are given, and tradeoffs discussed. The system was intended for teaching and research and no attempt was made to optimize performance. Nevertheless, timing measurements are presented to show that the use of a MUMPS-like tree structure for relational database system implementation can yield performance which varies with the size of the operand relations in a manner similar to previously reported relational database implementations.

I. INTRODUCTION

Among the most significant current concepts in data base technology is the relational model of data base management. The theory has been discussed by Codd [1-3] and several other authors [4, 6, 8] . The advantages of the relational model of data include simplicity, symmetry, data independence, and semantic completeness. Several implementations of the relational model have been reported in the literature [4,8] . The Data Manipulation Languages (DML) used in these implementations can be classified into six major groups: (1) Algebraic, (2) Relational Calculus, (3) Mapping-oriented, (4) Element-by-element, (5) Natural language, and (6) Graphic oriented. Because of its relative ease of understanding and use, a relational algebra DML was chosen for the present implementation. Although certain kinds of information structures cannot readily be represented as trees, it is shown here that the MUMPS* binary tree structure is suitable for the implementation of a relational database management system.

Within MUMPS, binary-tree structured files are referred to symbolically as multiply-subscripted arrays, known as global arrays, or simply globals. These globals (trees) are dynamic and sparse, i.e. the only nodes which are created are those to which a value is explicitly assigned or those which must be created in order to provide a path from the root to a node which has been assigned a value. Any node may contain either numeric or string data of variable types and lengths. Some MUMPS systems have facilities for preventing deadlocks and insuring error free concurrent access to the shared data base represented by the appropriate set of globals [5, 10]

The choice of MUMPS globals as the supporting structure for the MRDB system was motivated principally by the availability of a MUMPS system, the anticipated ease of implementation, and the interest in such work among the growing community of MUMPS users.

In the rest of the paper, we describe a prototype relational system implemented in DEC MUMPS-11 [5] , on a PDP-11 computer. Examples of operations will be given, and timing measurements presented.

2. OVERVIEW

It will be assumed in this paper that the reader is familiar with the basic concepts of relational algebra (References [1-4] should be helpful).

As viewed by the user, the database consists of a collection of named relations in normal form. These time-varying relations are of assorted degrees; and as time progresses, each n-ary relation may be subject to transformation, extraction, deletion, insertion, and alteration of some or all of its existing n-tuples.

For every data base there is a corresponding work-space of variable size. This workspace is intended to facilitate the user's manipulation of the database by providing him with a temporary storage space for an arbitrary number of transient or intermediate relations.

Both database and workspace grow and shrink dynamically as needed. They interact with each other as well as with the user through a set of commands, operators, and special functions which will be described later.

The MRDB Management system has the required facilities to insure a modest level of security and integrity of the stored data. It can be used as a stand-alone system or as a data management facility for the MUMPS language, with a flexible interface. The user interacts with the MRDB system through an algebra-based DML which will be described next.

3. The DML

*Massachusetts General Hospital Utility Multi-programming System

In its present state, the MRDB system operates either on individual queries or on a sequence of queries, as determined by the user and his application. Each query is made out of a series of commands and/or functions separated by the delimiter semicolon (;). Within the command itself, arguments are separated by the delimiter colon (:). The dollar sign ($) is the first of four characters that make up every function's name. Functions with arguments, and operators within commands, have their arguments enclosed within parentheses. These arguments are relations' names and/or the associated domains' names, separated by either the delimiter (,) or the delimiter (.) as will be described later. See fig. 1.

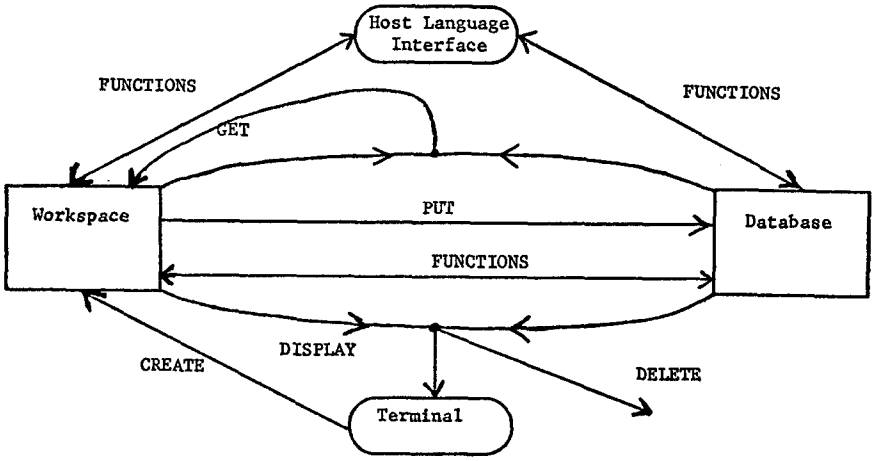


Fig. 1 - A simplified view of Interactions

Let the ␣ character represent the blank character, and Rj represent any relation which exists in the database and/or the workspace for j=1,2,3,... Let Dji represent any domain's name in relation Rj for i=1,2,3,... and let
$\theta \in \{N,=,<,>,(=<,<=),(=>,>=),(<>,><)\}$.

A. The five major DML commands have the following general forms:

CREATE ␣ R1:R2:R3:---
DELETE ␣ R1:R2:R3:---
DISPLAY ␣ R1:R2:R3:---
PUT ␣ R1:R2:R3:---
GET ␣ R1 ␣ OPR(ARG):R2 ␣ OPR(ARG):R3 ␣ OPR(ARG):---

a) The CREATE Command allows the user to enter, from the terminal into the workspace, relations under the names R1, R2,...
b) The DELETE command will delete the named relations from the workspace and/or the database, if such relations exist.
c) The DISPLAY command, upon execution, will display on the terminal the named relations in the respective order and in tabular form. Any of these relations may be in either the workspace or the database at the time of execution.
d) The PUT command will store the named relations in the database. These relations are assumed to exist currently in the workspace.
e) In the GET command, OPR and ARG stand for operator and argument(s) respectively. The operators are those defined by Codd [3] , and are abreviated by their first three letters in the DML of the MRDB system.

Every operator operates on its arguments to produce a resultant relation, and then stores this relation in the workspace under the associated name Rj. Any one of the resultant relations may be used as a part of the argument(s) of any subsequent operators under the same GET command, or other following commands and/or functions. Table 1. briefly summarizes the role played by the "OPR(ARG)" in the GET command.

B. Some of the more commonly used functions in database queries have been implemented in the MRDB sytem.

These functions return their outputs to, or take their inputs from, a set of preassigned global arrays and/or variables, enabling a MUMPS application program to manipulate the given relations directly if desired. The relations in the arguments of the functions are assumed to be in the workspace and/or the database at execution time. Table 2 gives a brief description of the currently existing functions. Finally, we mention that those commands and functions which involve the alteration of data in the database will be prevented from being carried out by unauthorized users. In turn, authorized users are prevented from carrying out operations which would violate any of the system integrity constraints (see the appendix for examples).

4. IMPLEMENTATION

Each relational database is represented by a MUMPS global as shown in fig. 3. It is seen that the tabular form of a relation (fig. 2) is mapped into a sub-tree. In addition to the above structure, relations resulting from certain operations could be stored so that each tuple in the given relation would be directly addressable by means of a hashing function which transformed the tuple number (global index) into a unique relative address on the disk. This randomly addressable structure would be a useful adjunct to the normal tuple storage scheme in speeding up some of the relational operators.

No ordering among tuples, or among data items in a domain, is assumed.

The general implementation technique of MUMPS globals is to map logical information at a given subscript level of a given array into groups of fixed size blocks chained together linearly to contain all the data values stored at that subscript level and all the pointers which point to the headers of the chained blocks at the next lower subscript level. Logically, the global looks like an n-ary tree, but the physical implementation is that of a binary tree, with one real pointer from a parent node to the first descendent. In the following, a

Table 1 - Relational operators as used by the MRDB system

SYNTAX OF OPR(ARG)	CORRESPONDING MEANING
1. CAR(R1,R2)	1. The cartesian product of relations R1 and R2
2. UNI(R1,R2)	2. The union of relations R1 and R2
3. INT(R1,R2)	3. The intersection of relations R1 and R2
4. DIF(R1,R2)	4. The set difference of relation R2 from R1
5. PRO(Rj.Dj1.Dj2.Dj3.---)	5. Projection of Rj over Dji's. Dji's could be in any order. Redundant tuples will be removed from resultant relation.
6. JOIθ(R1.D1i,R2.D2j)	6. R1 is θ-joined on domain D1i with R2 on domain D2j.
7. RESθ(Rj.Dj1.Dj2)	7. Relation Rj is θ-restricted on domain Dj1 and Dj2.
8. DIV(R1.D11,D12,D13.---, R2.D21.D22.D23...)	8. Division of R1 on domains D11,D12,D13, --- by R2 on domains D21, D22, D23, --- If R1 is empty, the resultant relation will be empty too.

simple timing analysis for some relational operations will be carried out.

Given three memory buffers, b_1, b_2, and b_3, each capable of storing one disk block, assume that the highest global level (level 1 of fig. 3) is small enough to be stored in main memory. Let Ti,Di,Ki for i=1,2,... be the number of tuples, the number of domains, and the average number of data items of relation Ri that can be stored in one disk block. Let Nj be the number of disk blocks to be accessed in order to execute a command, an operator or a function; and let that Nj(j=1,2) be our measure of the speed of response to queries, i.e. the smaller the Nj, the faster the response to the given query. Almost all MUMPS applications systems are disk-bound, and thus Nj is a realistic measure of system speed.

Table 2 - The MRDB system special functions

FUNCTIONS SYNTAX	MEANING
a. $MAX(Rj.Dji)	Returns maximum value in domain Dji of Rj.
b. $MAS(Rj.Dji)	Returns longest data item in domain Dji of Rj.
c. $MIN(Rj.Dji)	Returns minimum value in domain Dji of Rj.
d. $MIS(Rj.Dji)	Returns shortest datum in domain Dji of Rj
e. $CNT(Rj.Dji)	Returns the number of distinct data items in domain Dji of relation Rj.
f. $TOT(Rj.Dji)	Returns the sum of data items in Dji of Rj.
g. $AVR(Rj.Dji)	The average value of data items in Dji of Rj.
h. $ORA(Rj.Dji)	Orders the tuples of Rj in ascending order on domain Dji.
i. $ORD(Rj.Dji)	Same as $ORA but in descending order.
j. $CNW(R1,R2)	Changes the name of a workspace R2 to R1.
k. $CND(R1,R2)	Changes the name of a database R2 to R1.
l. $NOC(Rj)	Returns the number of domains of Rj.
m. $NOR(Rj)	Returns the current number of tuples of Rj.
n. $LOD(Rj)	Extracts Rj from database/workspace and return it to a designated global as in Fig. 3 when the MRDB is used as a sub-system to MUMPS.
o. $STR(Rj)	Works in the opposite way of $LOD(Rj).
p. $FDD	Returns a formatted list of currently available relations in the database with their associated domains.
q. $FDW	Does the same thing for workspace as in $FDD.

Many current relational database implementations treat the tuple as being the smallest retrievable unit of data in a stored relation. These structures will be referred to below as type 2. Let Ki* be the average number of data items of a relation Ri that could be stored in one disk block for a type 2 implementation.

In addition to retrieving data by tuples, the MRDB system implementation (referred to below as type 1) offers the capability of accessing relations by domains and/or data items selectively. This extra feature reduces the I/O time (Nj), at the expense of extra storage and more system overhead ($Kj^* \geq Kj$), by allowing only the necessary data (required domains) to be transferred between the disk and main memory. In the rest of this section the performance of type 1 and type 2 structures for some relational operations will be briefly compared.

Consider first the DISPLAY and CREATE commands. Since these commands require the transfer of a whole relation to and from the memory buffers, the minimum possible values of Nj are $N_1=DiTi/Ki$ and $N_2=DiTi/Ki^*$ for type 1 and type 2 respectively.

EMP:

NAME	SALARY	MANAGER	CODE
John	20,000	Jones	J25
Georges	3,000	Thomas	G16
Lee	7,260	Susie	L22

Fig. 2 - A tabular form of relation EMP

Since we considered level 1 of fig 3 to be in main memory, we find that deletion of a whole relation (by the DELETE command) requires no disk access, i.e. $N_1=0$. This is very fast, because all that is required is deletion of a single pointer.

It should be mentioned that, in theory, the PUT Command should require no more than a single pointer change, and thus $N_1=0$. However, because of the way globals are implemented in the present DEC MUMPS-11 system, the PUT Command requires the reconstruction of an entire sub-tree.

Finally, we show, through simple formulas, the amount of time needed to carry out some of the functions from table 2 and the PRO, JOIθ, and RESθ operators.

4.1 Functions

For a type 1 implementation it takes $\lceil Di/2Ki \rceil$ ($\lceil x \rceil$ means the smallest integer $M \geq x$) blocks to locate the given domain and $\lceil Ti/Ki \rceil$ blocks to transfer all data items in the selected domain.

$$N_1 \approx \lceil (Di+2Ki)/2Ki \rceil .$$

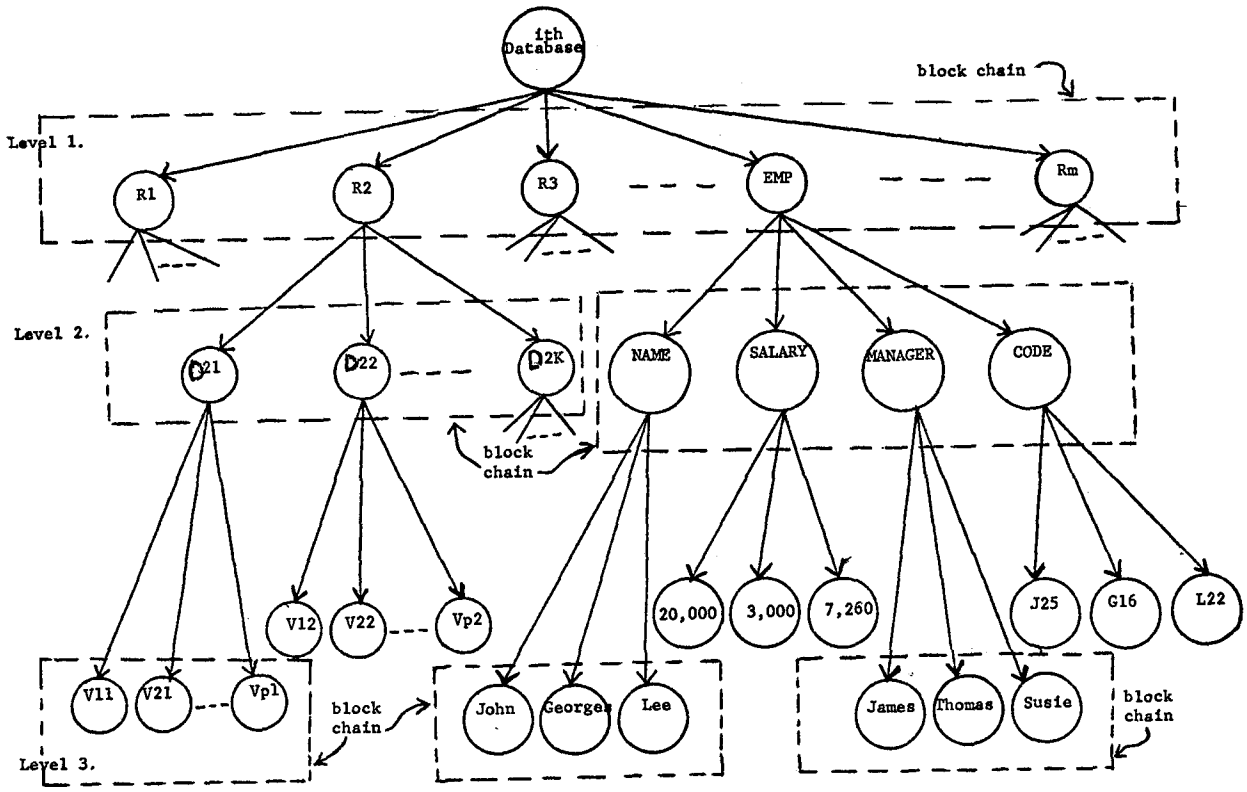
For type 2 implementations, it takes $N_2 \approx \lceil DiTi/Ki^* \rceil$ blocks to carry out the the function. $\Rightarrow$

$N_1/N_2 \approx ((Di+2Ki)/2Ki)(Ki^*/DiTi)$

If we assume $Ki^* = \alpha Ki$ $(1 < \alpha \leq 2)$ and $Di << 2Ti$ $\Rightarrow N_1/N_2 \approx \alpha/Di$ which says that response with a type 1 structure will be Di/2 to Di times faster than with a type 2 structure.

4.2 Projection (PRO)

To carry out the projection operation, all that is necessary is to cut the links leading to the descendent nodes (of the appropriate domains) from



N.B. 1) Vij refers to data item in tuple i and domain j for i = 1,2, --- p and j = 1,2, ---k

2) A block chain consists of one or more disk blocks linked by continuation pointers.

Fig. 3 - RDB Structure in MUMPS Global form

the parent node (the operand relation). This requires retrieval of the appropriate blocks, erasure of the pointers, and storage back on the disk. Thus, the value of N_1 is in the range $2 \leq N_1 \leq \lceil Di/Ki \rceil + 1$ depending on the number of projected domains and their locations. This is very fast in comparison with other implementations [8].

4.3 Join (JOIθ)

Let S be the number of tuples of the result relation. $\Rightarrow$

$N_1 \leq \lceil D_1/2K_1 \rceil + \lceil D_2/2K_2 \rceil + \lceil (T_1/K_1)(T_2/K_2) \rceil + S$

In the typical case, $Ti \gg Di$, we can ignore the first two terms, since they will be negligible compared with the values of the last two terms. It is easily seen that $N_2 = \lceil D_1T_1D_2T_2/K_1{*}K_2{*} \rceil$

$\Rightarrow N_1/N_2 = (T_1T_2+K_1K_2S)K_1{*}K_2{*}/D_1D_2K_1K_2T_1T_2$

If we assume $K_1{*} = \alpha_1K_1$ and $K_2{*} = \alpha_2K_2$, then

$N_1/N_2 = 1/P = \alpha_1\alpha_2(T_1T_2+K_1K_2S)/D_1D_2T_1T_2$

This says that as long as $K_1K_2S < (D_1D_2 - \alpha_1\alpha_2)T_1T_2/\alpha_1\alpha_2$ is true, the Join operation with a type 1 structure is P times faster than with a type 2 structure.

As a simple example, let $D_1=10$, $D_2=12$, $T_1=200$, $T_2=150$, $K_1=14$, $K_1{*}=20$, $K_2=8$, $K_2{*}=14$, and $S=70$. We see that the Join operation is about 38 times faster when using a type 1 structure.

4.4 Restriction (RESθ)

It can be shown here that $N_1 \leq \lceil Di/2Ki \rceil + \lceil 2Ti/Ki \rceil + S$ using the same reasoning as in section 4.3, we can neglect the first term in the previous formula. For type 2, we have $N_2 = \lceil TiDi/Ki{*} \rceil$.

$\Rightarrow N_1/N_2 = 1/P = (2Ti+KiS)\alpha/TiDi$; which says that, as long as $KiS < (Di-2)Ti/\alpha$, the Restriction operator when using a type 1 structure is faster by P times than when using a type 2 structure.

For example, for the values Ti=400, Ki=10, S=60, Di=10, and α=1.5, we get $P \approx 2$.

4.5 Measurements

Finally, table 3 displays a comparison between the measured and predicted relative speeds for certain operations. Note that the time units have been normalized in order to show the change in speed with changes in the size of the relations.

Table 3.a - Functions operating on relations with a fixed number of tuples

number of domains	1	2	3	4	5	6	7	8
measured time	1	1	1.12	1.37	1.37	1.5	1.6	1.75
predicted time	1	1	1.1	1.2	1.25	1.4	1.5	1.58

Table 3.b - Functions operating on relations of a fixed degree

number of tuples	5	10	15	20	25	30	35	40
measured time	1	1.33	1.55	1.77	2	2.55	2.77	3.33
predicted time	1	1.3	1.5	1.8	2	2.3	2.6	3.15

Table 3.c - PRθ operation on a relation of degree 9 with 30 tuples

number of projected domains	1	2	3	4	5	6	7	8
measured time	1.46	1.46	1.3	1.15	1.15	1.07	1	1
predicted time	1.4	1.4	1.35	1.35	1.1	1.05	1	1

Table 3.d - RESΘ operation on relations with varying degrees and a fixed number of tuples.

number of domains	2	3	4	5	6	7	8	9
measured time	1	1.25	1.37	1.62	1.87	2	2.25	2.25
predicted time	1	1.2	1.35	1.6	1.82	2	2.3	2.35

5. CONCLUSIONS

An algebra based, relational database management system implemented with hierarchical (binary tree) files has been presented. The available operations have been discussed, and the retrieval/update facilities demonstrated through a set of simple examples. The implementation has been briefly discussed, and some experimental measurements compared to predicted values for speed of response.

The system is adequate for educational purposes, and capable of yielding useful data on file structure performance. Further use of the system presented here for the investigation of file structures in relational database management systems is planned. In particular, the present MRDB system will be used to test a parameterized theoretical model of DBMS performance.

APPENDIX

In order to familiarize the reader with the MRDB system, we are going to work out in this appendix a few simple examples. Each example will consist of a request, worded in English, and a corresponding solution, expressed by the DML queries. Each solution will be underlined.

First, assume that we have information stored about suppliers, parts, and projects as expressed by the following four relations [4] .

S(S#,SNAME,CITY,STATUS)
P(P#,PNAME,COLOR,WEIGHT)
J(J#,JNAME,CITY)
SPJ(S#,P#,J#,QTY)

where a tuple in relation S gives information about a supplier with a given unique ID, his name, location, and status. A tuple in relation P tells us the part number, name, color, and weight. A tuple in relation J tells us the project number, its name, and location. Finally, the significance of an SPJ tuple is that the specified supplier (S#) supplies the specified part (P#) to the specified project (J#) in the specified quantity (QTY).

The examples follow (refer to fig. 4 for relations C1, C2, C3, C4, C5 and C6).

R1: Display full details of all projects. Add some new tuples, update the J relation, then store it in the data base.
S1: DISPLAY␣J;CREATE␣NEW;GET␣J␣UNI(J,NEW);PUT␣J
R2: Get full details of all projects in London, then display full details of all projects that are not in London.
S2: CREATE␣C1;GET␣IN␣JOIN(J.CITY,C1.A):OUT␣DIF(J,IN);DISPLAY␣OUT
R3: Get S# values for suppliers who supply project J1, then the supplier with the shortest name.
S3: CREATE␣C2;GET␣W␣JOI=(SPJ.J#,C2.A):W␣PRO(W.S#);$MIS(S.SNAME)
R4: Display a formatted list of all currently existing relations, then get S# values for suppliers who supply project J1 with part P1.
S4: $FDD;$FDW;GET␣M␣PRO(SPJ.S#.P#.J#):G␣DIV(M.P#.J#,C3.P#.J#)
R5: Get P# values for parts supplied to any project in London.
S5: GET␣K␣JOIN(J.CITY,C1.A):CITY␣JOIN(K.J#,SPJ.J#):R␣PRO(CITY.P#)
R6: Display P# values for parts supplied to all projects in London.
S6: GET␣K␣PRO(K.J#):D␣PRO(SPJ.P#.J#):W4␣DIV(D.J#,K.J#);DISPLAY␣W4
R7: Display the J# values for projects supplied with at least all parts supplied by supplier S1, then change the name of relation S to SSS.
S7: 1. GET␣W1␣JOIN(SPJ.S#,C5.A):W2␣PRO(W1.P#):D␣PRO(D.J#.P#)
2. GET␣W4␣DIV(D.P#,W2.P#);DISPLAY␣W4;$CND(SSS,S)
R8: Get S# values for suppliers who supply the same part to all projects.
S8: GET␣M␣PRO(SPJ.S#.P#.J#):A␣PRO(J.J#):B␣DIV(M.J#,A.J#):B␣PRO(B.S#)
R9: Destroy some of the unneeded relations, then get S# values for suppliers who supply both projects J1 and J2.
S9: DELETE␣W4:W5:SSS;GET␣QTY␣PRO(SPJ.S#.J#):QTY␣DIV(QTY.J#,C6.J#)

C1(A)	C2(A)	C3(P# J#)		C4(A)	C5(A)	C6(J#)
London	J1	P1	J1	Red	S1	J1
						J2

Fig. 4 - Supplementary Relations.

REFERENCES

[1] Codd, E.F., A Relational Model of Data for Large Shared Data Banks, Communication of the ACM, vol. 13, no. 6, June, 1970.
[2] Codd, E.F., Normalized Data Base Structure: A Brief Tutorial, proc. of the ACM-SIGFIDET Workshop on Data Description, Access and Control, 1971.
[3] Codd, E.F., Relational Completeness of Data Base Sublanguages, Data Base Systems, Courant Computer Science Symposia Series, vol.6, Prentice-Hall, Englewood Cliffs, N.J., 1972.
[4] Date, C.J., An Introduction to Data Base Systems, Addison-Wesley, Reading, Massachusetts, 1975.
[5] Digital Equipment Corp., MUMPS-11 Programmer's Guide, DEC-11 MMPGA-C-D, Maynard, Mass., Nov.1974
[6] Held, Gerald Davis, Storage Structure for Relational Data Base Management Systems, Electronics Research Laboratory memorandum no. ERL-M533, the University of California, Berkeley, August, 1975.
[7] Knuth, D.E., The Art of Computer Programming: vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.
[8] Pecherer, Robert., Efficient Retrieval in Relational Data Base Systems, Electronics Research Laboratory memorandum no. ERL-M547, the University of California, Berkeley, October, 1975.
[9] Rotwitt,Theodore et.al.,Storage Optimization of the Tree Structured Files Representing Describtor Sets, proc.of the ACM-SIGFIDET Workshop on Data Description, Access and Control, 1971.
[10] Wasserman,A.I.et.al., MUMPS Globals and their Implementations, Available from the National Bureau of Standards, Room A-247, Building 225, Washington, D.C. 20234