RELATIONAL QUERIES IN A DOMAIN BASED DBMS

M. MISSIKOFF
IASI - CNRS
Rome - Italy

and

M. SCHOLL
INRIA
Le Chesnay
France

ABSTRACT

This paper addresses the problem of relational queries processing in the domain based database machine DBMAC. A brief description of the storage organisation of the domain based DBMS is first given. Then the operating principles of the domain based data-model, called D-model, is described through some examples. The central part of the paper deals with the translation of relational queries into operations on objects of the D-model. Objects of the D-model and a set of operations on these objects are first defined : this set S of operations is shown to be complete in that any relational query can be translated into D-model operations belonging to S.
Finally we give a method for processing relation queries using D-model operations.
The basic advantages to be expected from a domain based physical organization of data are :
1. fast equi-join execution,
2. a compact representation of intermediate results.
The latter should lead to efficient processing of complex queries, provided a powerful parallel physical architecture is chosen for implementation.

1. INTRODUCTION

In order to overcome the performance limitations of conventional Database Management Systems (DBMS), Database Machines (DBM) have been designed [1 to 10] endowed with the following characteristics :

(i) High speed (i.e. high throughput rate and low response time). In order to satisfy this high speed objective a Multiple Instruction Multiple Data (MIMD) [17] multiprocessor archirecture is a natural choice.

(ii) Data filtering : An "on the fly" filtering capability allows to speed up data access by scanning and processing it during its transfer from mass memory to main memory [5, 7]. A special purpose device (filter) located close to the disk device is capable of performing a significant part of data manipulation at the same speed as the disk transfer.

(iii) Physical Data Storage Organization : Choose adequate data structures (using indexing, linking, clustering partitioning) in order to minimize 1) the net amount of data accessed by a transaction, 2) the overhead incurred by updating, 3) conflicts due to multiple access. Objective 1) is usually contradictory to objectives 2) and 3) and some tradeoff must be chosen in order to meet these 3 objectives.
The DBMAC domain based data organization [14, 15] is an attempt to meet the above characteristics. This organization is obtained by performing an attribute partitioning of the relations of the database followed by a reaggregation of the attributes defined on the same domain. The database then is represented by a collection of domains.

The implementation of each domain representation, called Data Pool [15], has an organization typical of inverted files. The peculiarity of the proposed data organization is that this inverted structure does not include pointers to base relations : the Data Pool is the only data aggregation of the database and stores all the data values.

Finally, the intermediate results, produced during the execution of a query, are kept as files of tuple identifiers (tids) : each tid represents a reference to a given tuple of a given relation.

This paper addresses the problem of relational queries processing with such a domain based organization.

An informal description of the data organization is first given by means of a few examples (sections 2 and 3). We emphasize the behaviour of the proposed data model when retrieval operations are performed. The structure used for implementing the Data Pools was described in [14] as well as the low level primitives and a first performance evaluation.

We then define in section 4, more specifically, new objects called D-objects as a representation of the domains and a set S of operations on D-objects is given. This set S is shown to be complete, i.e. we show that any relational algebra operation can be translated into a subset of operations of S on D-objects (section 5).

We finally give a method for processing relational queries using D-model operations (section 6).

2. STORAGE ORGANIZATION

A Data Pool (DP) includes all the attribute values (without duplicates) defined on a given domain that actually appear in the database.

It is therefore necessary to keep track for each value of the DP

- to which tuple of which relation it belongs
- to which attribute it belongs (several attributes may be defined on th same domain in a given relation and/or in different relations).

To this end, the DP contains some additional control information.

We now consider a single domain and the structure it assumes in order to correctly hold values and relational entities to which they belong.

We define a domain tree having constant height 5. The root (level D) contains the name of the domain ; at the other four levels we have :

level 1. : node **v** - contains a **value** of the domain
level 2. : node r - contains a **relation** identifier
level 3. : node a - contains an attribute identifier
level 4. : node t - contains a tuple identifier (TID)

Each path from the root to the leaf :

$$p(d,v,r,a,t)$$

identifies an instance of an attribute.

On the figure below a domain tree is represented :

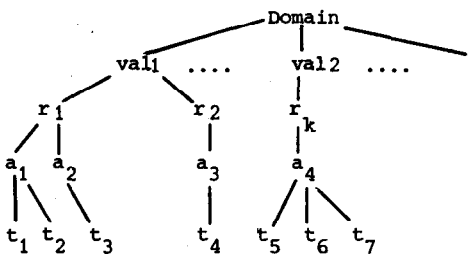


Fig. 1. Domain tree

For the sake of simplicity, we assume that in a given relation, two attributes cannot be defined on a **same** domain. This implies the elimination of level 3.
For further details see [15].

The domain tree is implemented in DBMAC by means of an inverted file organization. This kind of data organization has been adopted in others existing projects [11, 12]. However, in such projects, the inverted files are some supplementary structures used to speed up retrieval operations while flat files are also kept to store base relations. Moreover, the intermediate results, produced in computing a multistep relational query, have the same format as base relations.

In DBMAC DP's are the only structures holding data : there are no tuples stored in flat files to form base relations. The intermediate results are represented as sets of tuple identifiers (tids). On the one hand, this representation is very compact, and permits to compute a significant part of the query with very simple operations (as intersection and union of tids), as we will see later ; on the other hand it is necessary to add a final step for the materialization of output tuples.

3. QUERY PROCESSING IN DBMAC

In this section, we describe the way a query is processed in the context of a domain based database organization. For the sake of clarity, we use examples of simple queries, using the Selection, Projection, Join (SPJ) subset of relational algebra operations. The more general case will be treated in section 5.

The SPEC database

Take the following database schema :

S - Student (SNAME,TOWN,AGE)
P - Professor (PNAME,TOWN,TEL)
E - Exam (COURN,SNAME,GRADE)
C - Course (COURN,ROOM,PNAME)

to which corresponds the domain oriented internal schema (Domains with attributes defined on it).

NAMES (S.SNAME,P.PNAME,E.SNAME,C.PNAME)
TOWNS (S.TOWN,P.TOWN)
COURSES (E.COURN,C.COURN)

The remaining attributes : AGE,TEL,ROOM,GRADE are defined on four disjoint domains having the same names as the attributes.

The figure below illustrates two instances of the relations STUDENT and PROFESSOR and the common domain TOWNS.

	SNAME	TOWN	AGE
1	LULU	PARIS	20
2	TOTO	ROME	21
3	BIBI	PARIS	21
4	MIMI	PISA	19

	SNAME	TOWN	TEL
1	LILLI	PARIS	19
2	BETTY	PARIS	11
3	NICO	ROME	15
4	MISHA	ROME	16

RELATIONS STUDENT AND PROFESSOR

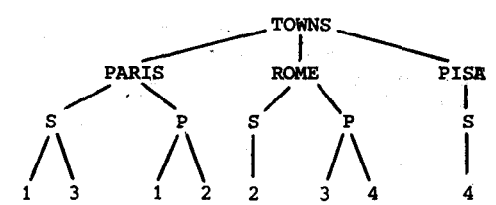


Fig. 2. Domain TOWNS

Note that during the transformation of a database into a domain based organization we need a numbering function to produce the tids.

As an example we shall give two queries on this database. Let us first show informally how the SPJ relational operators are computed in our model. A more formal definition is given in Section 4.

Selection DSEL ($\sigma_{C(A,R)}$) : This operator selects in relation R the tuples that satisfy condition C on attribute A. It is performed by scanning the domain tree on which A is defined and checking for C on the values. Each time that a value V satisfied C we check whether R is a son of V ; if yes we add the corresponding tids to the result.

Equijoin DJOIN ($R *_D S$) : This operator performs the equijoin on attribute A (of R) and attribute B (of S) defined on the same domain D. It is performed by scanning the domain tree. Each time R and S are both sons of the same value V, the cartesian product of the tids linked to R and S is added to the result.

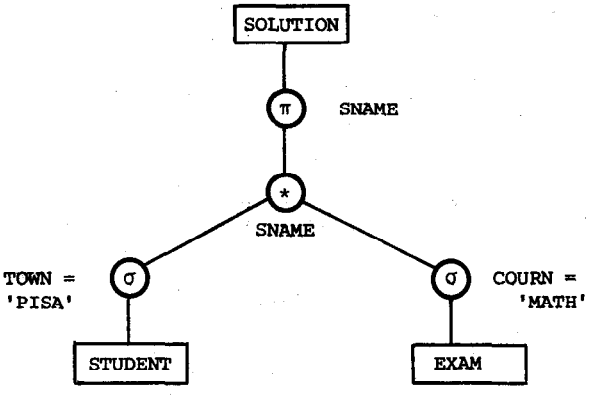
Project DPROJ ($\pi_{X,R}$) : This operator projects relation R on the subset X of attributes. Since the intermediate results of relational operators are lists of tids and since each list of tids can represent the whole tuples or only some of its attributes, the project is not an operation that affects the tids representation.

Therefore, in our model, the project operation is not taken into account until the last step of the query, when it can be used to materialize the tuples of the answer to be given back to the user.

Using these three operators we can formulate the queries first expressed in natural language and then represented by a query tree.

Q.1. : "Find the names of students living in PISA and having taken the MATH exam"

The following SPJ query tree satisfies the query :



The square leaves represent the relations on which the select (σ) operators are applied. The σ nodes are labelled by the selection condition, the * node (equijoin) is labelled by the JOIN attribute and the π node (projection) is labelled by the projection attribute.
Query Q.1 operates on three attributes : TOWN, COURN, SNAME.

With a domain based organization the solution is obtained by scanning the corresponding domains to produce the first intermediate results as sets of tids. Then these tids sets are progressively joined and projected. The output is finally materialized from the resulting set of tids and the Datapools corresponding to the attributes to be projected.

In the figure below two equivalent trees (referred to as D-trees) that compute query Q1 in a domain based database are presented (output materialization has been omitted).

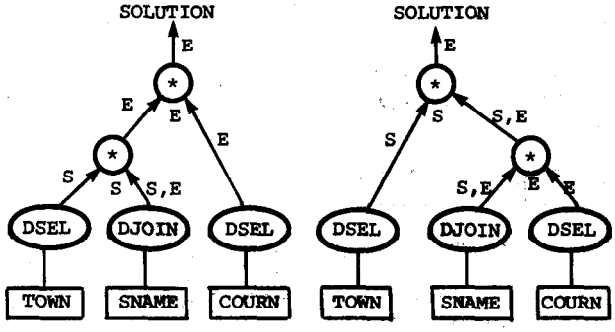


Fig. 3. D-trees that compute Q_1

The square leaves represent the domains involved in the query. The circular nodes just above (called production nodes) represent the only operations (DSEL or DJOIN) that are performed directly on the domains. They produce tid sets that flow along the edges.

As an example, Selection DSEL on domain TOWN produces a set of tids of relation S, while Equijoin Djoin on domain SNAME produces a set of tids S, E identifying tuples of relation S * E (Join of Relation S and Relation E or attribute SNAME).

The tids sets that flow up are successively joined (and projected) in the remaining nodes, called reduction nodes and labelled with the relation to which belong the tids on which the join is performed.

As an example in the left D-tree at level 2 S-tids are joined with S, E tids (join attribute : S-tid); We keep only the E-tids (projection on E-tids).

Q.2. : "Find the telephone of the professor that examined students younger than 21 and rated with a grade greater than C".

An SPJ query tree for Q_2 in a classical DBMS would be the following :

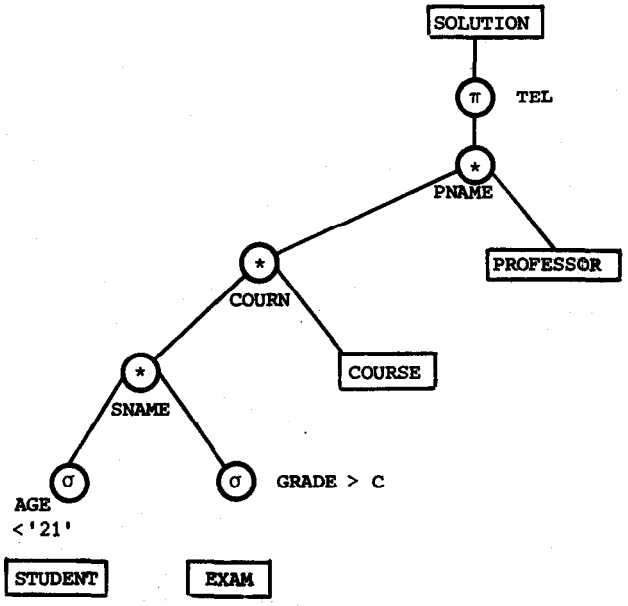


Fig. 4. An SPJ query tree that computes Q_2

A D-tree that computes Q_2 is the following (without output materialization) :

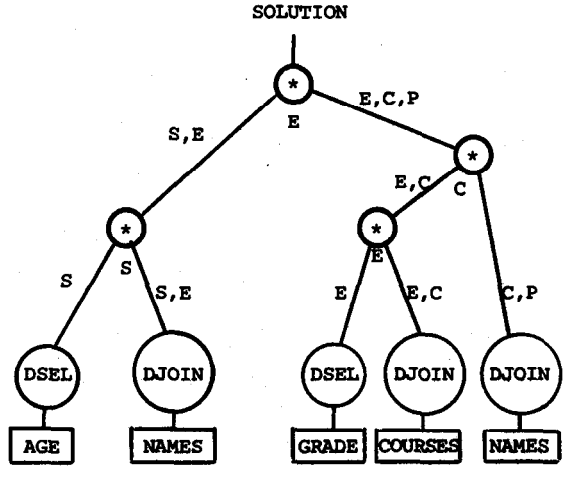


Fig. 5. A D-tree that computes Q_2

The fan-out of the tree depends on the number of domains involved in the query.

The execution of the tree permits a high degree of parallelism ; in fact the system can start to scan in parallel all the attributes. Then, as soon as the required intermediate results are ready, the above tids joins can be performed.

In the above examples, three D-operations are necessary on two D-objects : DSEL and DJOIN were used on the Data Pool objects, while tids joins was performed on tids sets.

In the following section, we define more precisely the D-model, i.e. D-objects and D-operations on these objects and show in Section 5 that any relational expression referred to as R-expression can be translated into a D-expression (whose operators and operations are those defined in the D-model).

4. THE D-MODEL

4.1. D-objects. The first object is the Datapool. Given a Relational schema :

$$R(A1:D1, A2:D2, \ldots, An:Dn)$$

The Datapools are the relations

$$D_i\ (V,R,A,T),\ i \in \{1,n\}$$

where D_i is the i-th Domain identifier

- V represents the values of the domain actually stored in the database
- R is the identifier of the relation defined on that domain :
- A is an attribute name of the relations defined on that domain
- T represents the tids.

A value of the Datapool for domain TOWNS is given below :

- TOWNS -

V	R	A	T
PARIS	S	TOWN	1
PARIS	S	TOWN	3
PARIS	P	TOWN	1
PARIS	P	TOWN	2
ROME	S	TOWN	2
ROME	P	TOWN	3
ROME	P	TOWN	4
PISA	S	TOWN	4

Given the collection of Datapools values it is possible to obtain the original relation r value as follows :

(i) for each attribute ai, i ∈ (1,n) of the target relation r select Dj (on which a_i is defined) on R and A and project on V and T :

$$IR_1(V,T) = (Dj \mid R=r,\ A=ai)[V,T]$$

(ii) then perform the join on T for all the intermediate results and project on values :

$$r = (IR_1 *_T IR_2 *_T \ldots *_T IR_n)\ [V1,V2,\ldots,Vn]$$

Derived Relation

We have so far introduced one D-object. The second type of object called Derived Relation (DR) represents intermediate results of a query execution.

A DR is a relation

$$DR(R_1,T_1,R_2,T_2,\ldots,R_p,T_p)$$

whose attribute R_i is a relation name and attribute T_i identifies tuples of relation R_i. Intuitively in the intermediate result (temporary relation), a t-uple is constructed from a tuple of relation R_1, a tuple of relation R_2,..., a tuple of relation R_p.

This representation turns out to be a rather compact way of representing temporary relations.

4.2. D-operations

The set of operations to be performed on D-objects can be subdivided in two parts :

1) Production operations - There are only three operations (two are referred to in Section 3 as DSEL and DJOIN) whose operand is a Datapool and whose result is a derived relation DR :

$$DR(R_k,T_k) = DSEL(D_i)$$

$$DR(R_k,T_k,R_\ell,T_\ell) = DJOIN\ (D_i)$$

$$DR(R_k,T_k) = DSALL(D_i)$$

DSEL and DJOIN can be defined as the following relational algebra expressions :

DSEL : $DR(R,T) = (D_i \ / \ R{=}r_k \wedge A = a_k \wedge V\ \theta\ v)[R,T]$

This is a restriction/projection on Datapool D_i (V,R,A,T) where we project on attributes R and T, and where the restriction condition expresses that attribute a_k from relation r_k must have a value which is θv (θ is an arithmetic comparison operator).

Clearly the resulting DR represents the subset of tuples of r_k satisfying the elementary condition $a_k\ \theta\ v$.

DJOIN : $DR[R_k,T_k,R_\ell,T_\ell] =$

$$(D_i \underset{V}{*} D_i \ / \ R_k{=}r_k \wedge R_\ell{=}r_\ell \wedge A_k{=}a_k \wedge A_\ell{=}a_\ell)\ [R_k,T_k,R_\ell,T_\ell]$$

This expression is a self-join of Datapool D_i on attribute V, followed by a restriction and a projection on attributes R_k, T_k, R_ℓ, T_ℓ.

Clearly the resulting DR represents the equi-join of relation r_k with relation r_ℓ on the attributes a_k and a_ℓ both being defined on the same domain D_i.

DSALL : $DR(R,T) = (D_i \ / \ R{=}r_k)[R,T]$

This selection/projection is a particular case of DSEL which produces in DR, the set of tuples identifiers of a relation.

Observe that in both DSEL, DSALL and DJOIN operations, we carry in the resulting DR, the relation identifier. Indeed both the relation identifier and the tid are necessary to uniquely identify a tuple of a given relation.

2) Reduction operations

The remaining D-operations are the four following binary relational operations applied to DR's : union, set-difference, Cartesian product and intersection. These operations take two DR's as operands and obviously produce a DR as a result.

Intuitively, once a DR has been produced (applying a production operator to a Datapool), the query execution continues on DR's using the four above operations until the final result is obtained. Observe we do not need the basic unary relational operations : Projection and Selection on DR's.

3) Output materialization

Once the first DR has been obtained, some "projection" process is necessary, i.e. an output operation which takes as an input a DR and one or several Datapools materializes for the user, the output as a traditional file of tuples values.

Without loss of generality, assume the final result is the DR

$$DR(R_1,T_1,R_2,T_2)$$

where $R_1{=}r_1$ and $R_2{=}r_2$, and the user is only interested in the projection of attribute a_1 of relation r_1 and attribute a_2 of relation r_2. Assume a_1 and a_2 are respectively defined on domains D_1 and D_2.

The following expression materializes the solution as an output relation OR :

$$OR[V_1,V_2] = (\{(D_1|A{=}a_1)[V_1R_1T_1] \times (D_2|A{=}a_2)\ [V_2R_2T_2]\} \underset{R_1T_1R_2T_2}{*} DR[R_1T_1R_2T_2])[V_1,V_2]$$

This expression can be decomposed as follows :

1. Select D_i on $A{=}a_i$ and project on V, R, T
2. Perform the Cartesian Product of the two intermediate results.
3. Perform the equijoin of the intermediate result with the DR on R_1, T_1, R_2 and T_2
4. Project on the values attributes V_1 and V_2.

5. COMPLETENESS

The purpose of this section is to show that the D-objects defined above and the set of D-operations on these objects so far introduced is complete. Given a database including a set of relations R_j, $j \in \{1,m\}$, we want to show that any relational expression whose operands are relations of the database can be expressed as a D-expression whose operators are D-operations and operands are D-objects. For this, we first define from the set of relations in the database, a set of datapools, D_i, $i \in \{1,n\}$ representing the domains on which the relations are defined. It is then enough to consider a set of primitive operations of relational algebra and to show how each of them is computed in the D-model. Following [16], we consider the

following primitive operations : union, set difference, cartesian product, projection and selection, Projection will be omitted below since we have assumed that it is not necessary for the query computation and have shown how an output materialization looks like (section 4). If necessary projection can be computed as in Section 4.

5.1. Union : $r = r_1 \cup r_2$

r is represented by a DR computed through the following steps :

1. Take any domain say D on which r_i is defined and perform the DSALL operation :

$$DR_i[R,T] = DSALL[D_i]$$

2. Compute the relational intersection $r_1 \cap r_2$ represented by the DR $DR_{1,2}$. This is done through the two following steps :

   a) for each domain D_k, $k \in \{1,n\}$ on which r_1 and r_2 are defined, perform the DJOIN operation :

$$IR_k[R_1,T_1,R_2,T_2] = DJOIN[D_k]$$

   b) Intersect all the intermediate results (DR's IR_k, k=1,n) and project the result on R_1, T_1 (R_1=r_1 and T_1 identifies a tuple of r_1) :

$$DR_{12}[R,T] = \left(\bigcap_{k=1,n} IR_k[R_1,T_1,R_2,T_2] \right) [R_1,T_1]$$

3. Compute the union as :

$$DR[R,T] = DR_1[R_1,T_1] \cup DR_2[R_1,T_1] - DR_{12}[R,T]$$

5.2. Set-Difference : $r = r_1 - r_2$

1. Extract the set of tids of r_1, i.e. perform DSALL on a domain of r_1, say D_1 :

$$DR_1[R,T] = DSALL[D_1]$$

2. Compute the intersection $r_1 \cap r_2$ represented by $DR_{1,2}$ obtained through steps 2a and 2b of Section 5.1.

3. Compute the set difference as :

$$DR[R,T] = DR_1[R,T] - DR_{12}[R,T]$$

5.3. Cartesian product

1. Extract the sets of tids of r_1 and r_2

$$DR_1[R_1,T_1] = DSALL[D_1]$$

$$DR_2[R_2,T_2] = DSALL[D_2]$$

2. Compute the cartesian product :

$$DR[R_1,T_1,R_2,T_2] = DR_1[R_1,T_1] \times DR_2[R_2,T_2]$$

5.4. Selection

We want to compute $r = \sigma_F(r_1)$: r_1 is restricted to the tuples satisfying boolean condition $F = c_1 \omega c_2 \ldots \omega c_p$, where ω is a logical operator ($\wedge, \vee, \neg$) and c_i is of the form $a_i \theta v$. Let D_i be the domain on which a_i is defined.

1. For each c_i, apply the DSEL operator :

$$DR_i[R,T] = DSEL[D_i]$$

2. Compute the Formula

$$DR[R,T] = F'(DR_1[R,T] \omega' DR_2[R,T] \ldots \omega' DR_p[R,T])$$

F' is obtained by replacing in F c_i by DR_i and ω by the corresponding set operator ω' ($\cap, \cup, -$).

If the query is of the SPJ type (the relational expression includes only Selections, Projections and Joins), it can be shown that an implementation using the D-model and taking advantage of parallelism inherent to an MIMD architecture and the computational power of special purpose devices is performant [13, 15] : DSEL and DJOIN are first performed in parallel by single domain searches (production steps), then the derived relations are combined (one or several reduction steps : see examples of section 3 ; observe in these SPJ examples, the only D-operation necessary on DR's is a join-project operation (Equi-join and projection).

6. Query processing

We have shown above that a relational expression can be translated into a D-exprassion, whose processing brings the expected solution to the initial relational expression.

A D-expression can be represented by one or several equivalent D-trees (see for example Fig. 3).

The purpose of this section is to give a simple way of producing the set of D-trees from an initial SPJ relational expression.

Query optimization will of course decide which is the "optimal" tree, to be selected for execution. The problem of choosing the best D-tree is not a trivial one and it goes beyond the goals of this paper.

Q-table

A Q-table is a tabular representation of an SPJ relational query in which columns are all relation names appearing in the query while rows represent all the attribute names in the query. The intersection of row i and column j is marked if attribute i belongs to relation j.

The figure below gives the Q-table for query 2 (see Section 3).

	C	E	P	S
AGE				X
SNAME		X		X
GRADE		X		
COURN	X	X		
PNAME	X		X	
TEL			X	

Fig. 6. Q-table for Query 2

D-tree processing

We give below an algorithm which solves for the query once the Q-table has been built.

To each row corresponds a Datapool (domain on which the row attribute is defined) and a production operation performed on this Datapool. If there is only one mark in the row it is a DSEL operation , e.g. on Datapool Age, we perform the following operation X_1 : (Age$|_{R=S \wedge V < 21}$)[R,T] which produces derived relation DR_1 identifying t-uples of relation S.

We shall not talk in the sequel of the "Projection" attribute TEL (The last operation is output materialization : projection of the telephone number, see section 5).

If there are two marks in a row, the production operation to be performed on the row Datapool is DJOIN, e.g. on domain NAMES, the following operation is performed :

$$X_2 : (\text{NAMES} \underset{V}{*} \text{NAMES}|_{R_1 = S \wedge R_2 = E \wedge A_1 = \text{SNAME} \wedge A_2 = \text{SNAME}})[R_1, T_1, R_2, T_2]$$

which produces derived relation DR_2 identifying the tuples of the relational join S.SNAME = E.SNAME.

The five production operations for query 2 are listed below with the corresponding relational subexpressions

Production		Operation	Datapool	Relation Subexpression
X_1	:	DSEL	AGE	S.AGE < '21'
X_2	:	DJOIN	NAMES	S.SNAME = E.SNAME
X_3	:	DSEL	GRADE	E.GRADE > 'c'
X_4	:	DJOIN	COURSES	E.COURN = C.COURN
X_5	:	DJOIN	NAMES	C.PNAME = P.PNAME

Fig. 7. Production operations for Query 2

D-Graph

From the Q-table construct a graph called D-graph as follows : to each row corresponds a node labelled with the production operation X_i. Two nodes are connected if they share at least one relation-name : a Q-table column owns at least two marks. More precisely, to a column with k marks corresponds in the D-graph a complete k subgraph whose edges are labelled with the column name (relation name).

The D-graph for query 2 is represented below :

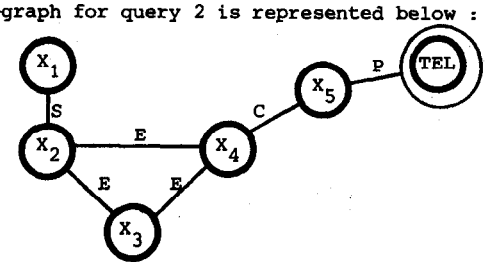


Fig. 8. D-Graph for query 2

Note that end node TEL corresponds to output materialization.

Algorithm

Clearly in the D-graph the nodes are the leaves (production nodes) of a D-tree (see Fig. 5).

Recall in D-tree processing, leave nodes are first processed, which steps produce sets of tids which are in turn combined.

The algorithm below which reduces the D-graph to one node corresponds to the complete query processing.

0. Take any spanning tree of the D-graph.

1. Process the Xi's.

2. Starting from a leaf of the spanning tree, repeat the following pairwise reduction until one node remains : a pairwise reduction corresponds to an elementary join between two derived relations.

3. Perform output materialization.

Reduction step

Reduce any terminal node of the spanning tree and its neighbor node n to one node and add the label

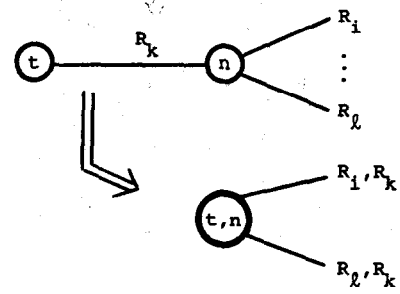


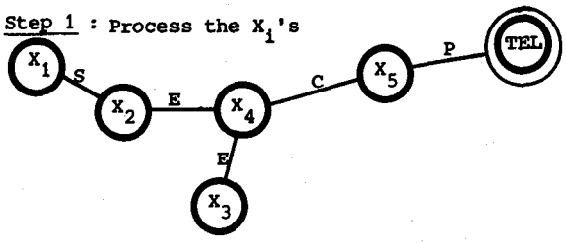
Fig. 9. Reduction step

of their common edge to all edges leaving n. This obviously corresponds to contracting two rows of the Q-table until its contains only one row with all columns marked.

Note that from the Q-table of Fig. 6 by a similar method one gets usual query trees such as that depicted in Fig. 4.

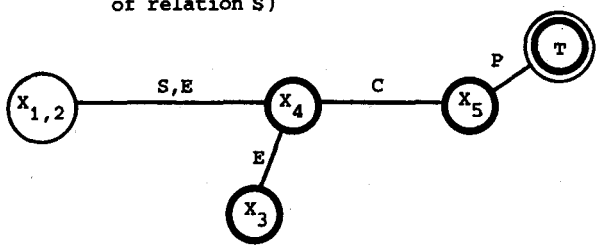
We give below an example of processing of Query 2. After each step, the reduced tree appears.

Step 1 : Process the X_i's



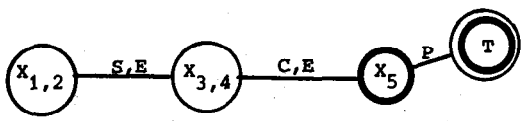
Step 2 : $X_{12} : DR_{12} = DR_1 *_{R_1,T_1} DR_2$

(Equi-join on R_1,T_1 that identify tuples of relation S)



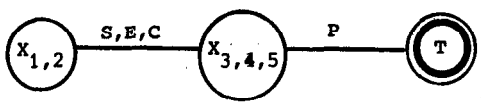
Step 3 : $X_{34} : DR_{34} = DR_3 *_{R_1,T_1} DR_4$

(R_1,T_1 identify t-uples of relation E)



Step 4 : $X_{345} : DR_{3,4,5} = DR_{3,4} *_{R_1,T_1} DR_5$

(R_1,T_1 identify t-uples of relation C)



Step 5 : $X_{12345} : DR_{1,2,3,4,5} = DR_{1,2} *_{R_1,T_1} DR_{3,4,5}$

(R_1,T_1 identify tuples of relation E)

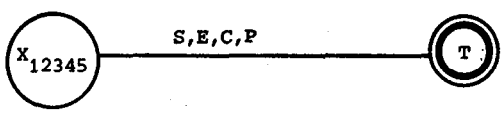


Fig. 10. Processing of query 2

The algorithm above corresponds to the execution of one one D-tree (Materialization has been omitted). Recall to a D graph it may correspond a large number of distinct D-trees, and for a given tree several steps may be run in parallel.

7. CONCLUSION

In this paper we presented a data model, called D-model, which allows a domain oriented representation of a relational database and showed that any relational expression can be translated into an expression of operations on the D-model (completeness with respect to the retrieval sublanguage). The proposed model should be particularly performant for certain classes of applications when implemented on a database machine [15].

In particular the effectiveness of the model is enhanced when the follwoing conditions are satisfied :

- the retrieval is limited to SPJ-relational expressions ;
- the DBMS based on the D-model is implemented on a parallel machine ;
- the parallel machine includes two special purpose devices for key operations :
  - (i) fast filtering of datapools to perform the production operations (i.e. DSEL, DJOIN) [13].
  - (ii) VLSI "intersector" to perform the reduction operations on tid sets.

The DBMAC group is currently investigating the feasibility of the above two devices and the possibility of having an optimiser for a fast translation of an input relational query tree into an optimal D-tree.

REFERENCES

[1] G.P. COPELAND, G.J. LIPOVSKI, S.Y.W. SU, "The Architecture of CASSM : A cellular System for Non-Numeric Processing", Proc. Ist Annual Symposium on Computer Architecture, Dec. 1973, pp. 121-128.

[2] C.A. OZKARAHAN, S.A. SCHUSTER, K.C. SMITH, "RAP-An Associative Processor for Data Base Management", Proc. 1975 NCC, Vol. 45, AFIPS Press, Montvale, N.J., pp. 379-387.

[3] H.O. LEILICH, G. STIEGE and H.Ch. ZEIDLER, "A Search Processor for Database management systems", Proceedings of VLDB, 1978, pp. 280-287.

[4] J. BANERJEE, R. BAUM, D. KHSIAO, "Concepts and Capabilities of a database Computer", ACM Trans. Database Systems, Vol. 3, N° 4, Dec. 1978, pp. 347-384.

[5] H. AUER et al. "RDBM - A Relational Database Machine" Information Systems, Vol. 6, N° 2, 1981, pp. 91-100.

[6] IDM 500 IDM Software Reference Mannual, Britton-Lee Inc, Los Gatos California.

[7] F. BANCILHON, M. SCHOLL,
"Design of a Backend Processor for a Database Machine",
Proc. of the ACM-SIGMOD Conference, Santa Monica, May 14-16, 1980, pp. 93-93g.

[8] D.J. DEWITT
"DIRECT - a Multiprocessor organization for supporting a Relational Database Management System",
IEEE Trans. on Comp., Vol. C28, N° 6, June 1979, pp. 395-406.

[9] G. GARDARIN et Al
"Objectifs Principes et Architecture d'une Machine Bases de Données Réparties",
Actes des Journées Machines Bases de Données, Sophia Antipolis, France, 10-12 Septembre 1980

[10] M. MISSIKOFF and M. TERRANOVA
"An overview of the Project DBMAC for a Relational Database Machine",
Proceedings of the 6th Workshop on Computer Architecture for Non-Numeric Processing,
Hyères, France, June 1981.

[11] CHUNG LE VIET et Al,
"Use of Abstracted Characteristics of Data in Relational Databases",
Proc. of COMPSAC , Nov. 79.

[12] SU, S.Y. et Al,
"MICRONET : A Microcomputer Network System for Managing Distributed Relational Databases"
Proc. of the 4th Int. Conf. on VLDB
Berlin 1978.

[13] MISSIKOFF M., SCHOLL M.,
"The Relational Database Machine DBMAC : FSA Filtering on a Fully invented Physical Organization of Data",
Proc. of the Working Seminar on Databases, Gressoney, Italy, Feb. 1982.

[14] MISSIKOFF M., TERRANOVA M.,
"The Architecture of DBMAC : A Relational Database Computer",
Proc. of the International Workshop on Database Machines, San Diego, Aug. 1982.

[15] MISSIKOFF M.,
"A domain Based internal scheme for Relational Database Machine",
Proc. of the ACM-SIGMOD Conf. Orlando, June 1982.

[16] ULLMAN,
"Principles of Database Systems",
Computer Science Press, 1980.

[17] KUNG H.T.,
"The Structure of Parallel Algorithms",
Advances in Computers, Vol. 19, 1980.