QUERY PROCESSING STRATEGIES IN THE PASCAL/R RELATIONAL DATABASE MANAGEMENT SYSTEM

Matthias Jarke
Joachim W. Schmidt*

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #33

GBA #82-28(CR)

This paper has been published in Proceedings 1982 ACM-SIGMOD International Conference on Management of Data, Orlando, Florida, June 1982, pp. 256-264.

*The work of Dr. Schmidt was supported in part by Deutsche Forschungsgemeinschaft, DFG, under grant no. SCHM 450/2-1.

QUERY PROCESSING STRATEGIES IN THE PASCAL/R RELATIONAL DATABASE MANAGEMENT SYSTEM

ABSTRACT:

In the database programming language PASCAL/R, the programming language PASCAL and concepts based on the relational data model are integrated. The paper investigates transformation strategies used in the PASCAL/R system to evaluate queries with existential and universal quantifiers. Intermediate data structures are described using a high-level language tool called a reference to a selected variable. The predicate calculus approach used in PASCAL/R offers new insight into recently proposed query optimization techniques and allows some of them to be extended.

1.0 INTRODUCTION

PASCAL/R [14] is a PASCAL-like programming language which offers a simple, structured, and uniform way to access a relational database [4]. In this paper, we present PASCAL/R's construct for data selection and some strategies for its efficient evaluation.

PASCAL/R's data definition and selection mechanisms generalize and combine well-known PASCAL concepts. The elementary set concept is extended to sets of structured elements (relations). Relation-valued expressions can be defined not only extensionally by enumeration of their elements, but also intensionally by free variables that range over relations and satisfy a selection predicate. Logical expressions are extended to first-order predicates by introducing existential and universal quantifiers.

The query evaluation strategies differ from most recently published work [2,3,18,19] in that they take into account universal quantification of variables. This has consequences for the handling of empty relations as well as for the generality of certain optimization techniques. A version of the system described here has been operational since spring, 1978 [7,8].

This paper is organized as follows. Section 2 describes first-order relational expressions in PASCAL/R. Section 3 introduces language constructs and intermediate data structures used in query evaluation. Section 4 presents strategies for efficiency-oriented query transformation. Section 5 reviews our results and gives directions for future research.

2.0 RELATIONAL EXPRESSIONS IN PASCAL/R

Figure 1 shows the declaration of a sample database in PASCAL/R. The four relations represent selected aspects of a computer science department: Employees, their recent publications, courses, and a (current) timetable that associates employees and courses. A RELATION can hold a variable number of identically structured elements. The elements are defined by component types (e.g., daytype, enumbertype) and are denoted by component identifiers (e.g., tday, enr). The list of component identifiers in angular brackets denotes the key.

A SELECTION is an intensional set definition used to compose a relation from other relations. It consists of two parts: A COMPONENT SELECTION specifies the components of the elements of the resulting relation. A SELECTION EXPRESSION specifies constraints on the relations contained in the component selection. Syntactical details can be

found in [17]. The following example illustrates the use of a selection.

EXAMPLE 2.1:

Assign to the unary relation, enames, the names of the employees of status professor who did not publish any papers in 1977 or who currently offer courses at a level of sophomore or lower.

```
enames:=
[<e.ename> OF EACH e IN employees:
 (e.estatus = professor)
 AND
 (ALL p IN papers
   ((p.pyear <> 1977) OR (e.enr <> p.penr))
  OR
  SOME c IN courses ((c.clevel <= sophomore)
   AND
   SOME t IN timetable
   ((c.cnr = t.tcnr) AND (e.enr = t.tenr))))]
```

As can be seen from the above example, the selection expression is a well-formed formula (wff) of an applied many-sorted first-order predicate calculus. Its atomic formulae are called JOIN TERMS; they can be either monadic (e.g., e.estatus = professor) or dyadic (e.g., e.enr = t.tenr). Any of the comparison operators =, <>, <, <=, >, >= may be used. Element variables (e.g., e) range over relations (e.g., employees) as declared in a RANGE EXPRESSION (e.g., e IN employees). These range-coupled variables can be free (EACH e IN employees), existentially quantified (SOME t IN timetable), or universally quantified (ALL p IN papers). A full definition of the calculus is given in [8].

A. Schmidt [13] proved that an expression of a many-sorted calculus can be converted into an equivalent one of a one-sorted calculus by introducing range expressions as another type of atomic formula and changing the original wff as follows:

Substitute

```
SOME rec IN rel (WFF)
by
SOME rec ((rec IN rel) AND WFF)
```

and

```
ALL rec IN rel (WFF)
by
ALL rec (NOT (rec IN rel) OR WFF).
```

Most of the well-known transformation rules for one-sorted predicate calculus [12] apply directly to the many-sorted case. The following lemma shows, however, two exceptions where empty relations may cause unexpected results.

LEMMA 1: Let A be a wff in which the variable, rec, does not occur and B any wff. Then the following four rules hold for the many-sorted calculus:

1. A AND SOME rec IN rel (B) =
   SOME rec IN rel (A AND B)
   (as in the one-sorted calculus)

2. A OR SOME rec IN rel (B) =
   - A, if rel = [] (the empty relation)
   - SOME rec IN rel (A OR B), otherwise

3. A AND ALL rec IN rel (B) =
   - A, if rel = []
   - ALL rec IN rel (A AND B), otherwise

4. A OR ALL rec IN rel (B) =
   ALL rec IN rel (A OR B)
   (as in the one-sorted case)

Proof: By transformation into one-sorted formulae.

Many systems evaluate queries directly as given by the user. We prefer a standardized starting point for optimization. Therefore, the PASCAL/R compiler transforms each selection expression into prenex normal form with a matrix in disjunctive normal form. It assumes that all range relations are non-empty but provides information to adapt the standard form at runtime if necessary.

EXAMPLE 2.2:

The statement in EXAMPLE 2.1 translates to

```
enames :=
[<e.ename> OF EACH e IN employees:
 ALL p IN papers
 SOME c IN courses SOME t IN timetable
 ((e.estatus=professor) AND (p.pyear<>1977)
  OR
  (e.estatus=professor) AND (p.penr<>e.enr)
  OR
  (e.estatus=professor) AND
   (c.clevel<=sophomore) AND
   (t.tenr=e.enr) AND (t.tcnr=c.cnr))].
```

If papers = [], this must be changed to

```
enames:=[<e.ename> OF EACH e IN employees:
                        e.estatus = professor].
```

In contrast, the above normal form would return the names of all employees.

In a query with only existential quantification, each conjunction of the standard form can be evaluated separately, because

SOME rec IN rel (WFF1 OR WFF2)

is equivalent to

SOME rec1 IN rel (WFF1) OR SOME rec2 IN rel (WFF2)

where WFF1 and WFF2 are any wffs. We show in section 4.3, below, that fully independent evaluation of conjunctions is not always desirable. In most queries with universal quantifiers, it is not even permitted, because the above separation transformation is not applicable.

3.0 QUERY EVALUATION FRAMEWORK

In this section, we outline a framework for query evaluation. First, we introduce a high-level language tool which allows one to manipulate element references (a generalization of TID's used in other systems) in PASCAL/R. We then discuss the data structures used in our algorithms. Based on these concepts, we describe a phase-structured procedure to evaluate standard form queries. In section 4, we investigate how to transform queries in order to reduce the processing effort.

3.1 Language Tools

In this subsection, a few more language constructs are introduced that can be used to describe the PASCAL/R query evaluation algorithms.

At first, a key-oriented selector mechanism, rel[keyval], is introduced by which so-called SELECTED VARIABLES can be named. This array-like notation selects elements of a relation, rel, by their key value, keyval, and makes them accessible as variables [17].

Based on selected variables, we introduce the notion of a REFERENCE, @rel[keyval], to a selected variable by which the access to a selected variable can be supported. A reference value can be stored in a variable of some newly introduced reference type which is denoted by prefixing the referenced relation variable by @. A selected variable can be regained from its reference by postfixing a reference variable by @. If we have an element variable, r, ranging over relation rel, we will often use @r as a short-hand for @rel[r.key]. More details of these language constructs can be found in [8,16].

EXAMPLE 3.1:

A primary index to the relation, employees, that associates key values and references is initialized and maintained by the following sketch of a program (:+ denotes the insert operator in PASCAL/R).

```
VAR employees : RELATION <enr> OF
                RECORD ... END;
    enrindex  : RELATION <enr> OF
                RECORD
                   enr  : enumbertype;
                   eref : @employees
                END;
 ...
BEGIN
 ...
 enrindex := [<e.enr,@e> OF
               EACH e IN employees: true];
 ...
 employees :+ [<20, technician, 'Highman'>];
 enrindex  :+ [<20, @employees[20]>];
 ...
END.
```

3.2 Data Structures

All intermediate structures described in this subsection are given in terms of PASCAL/R relations, mostly using references as components. Their implementation is discussed in [7,8].

A SINGLE LIST is a unary relation that stores references to relation elements satisfying a monadic join term. Similarly, an INDIRECT JOIN [11] is a binary relation that stores references to pairs of relation elements that satisfy a dyadic join term. Indirect joins are generated in two steps:

First, a (partial) INDEX on one relation involved in the join term is created. Next, the second relation is tested against the index. The first step can be omitted, if permanent indexes exist.

Figure 2 shows declarations for the single lists, indirect joins, and indexes of the join terms in EXAMPLE 2.2.

Let n variables v1, ..., vn occur in the selection expression. Relations with up to n components of type reference are used to store references to relation element combinations satisfying parts or all of the matrix of join terms.

3.3 Outline Of The Procedure

The structure of the evaluation technique outlined below is based on an early proposal by Frank Palermo [11].

In order to reduce working storage requirements, complete relation elements are replaced by references and no operation works on more than two relations. The final result is combined step by step from intermediate results representing the solution of subqueries. Therefore, the algorithm has a stage or phase structure:

1. The COLLECTION PHASE evaluates range expressions and single join terms. The results are single lists and indirect joins for all monadic and dyadic join terms in the selection expression. This phase performs data compression (records to references) and data reduction (testing join terms).

2. The COMBINATION PHASE manipulates only reference relations; it evaluates logical operators and quantifiers in three steps:

   Each conjunction is evaluated by combining the single lists and indirect joins obtained in the collection phase into n-tuples of references where n is the number of variables in the selection expression. This step involves operations like join or Cartesian product [6,9] of reference relations.

   The full disjunctive form is evaluated by a union operation on all these sets of n-tuples.

   If quantified variables occur in the selection expression, quantifiers are evaluated from right to left, using the relational algebra operations of projection for existential quantification and

division for universal quantification [5, 11].

3. The CONSTRUCTION PHASE dereferences the results obtained by the combination phase and projects on the components specified in the component selection.

EXAMPLE 3.2:

The subexpression of EXAMPLE 2.2,

```
(c.clevel <= sophomore) AND (c.cnr = t.tcnr)
```

is evaluated as follows: In the collection phase, a single list, sl_csoph, and an indirect join, ij_c_t, are created using an index, ind_t_cnr:

```
sl_csoph  := [<@c> OF EACH c IN courses:
                    c.clevel <= sophomore];
ind_t_cnr := [<t.tcnr,@t> OF
               EACH t IN timetable: true];
ij_c_t    := [<@c,t.tref> OF
               EACH c IN courses,
               EACH t IN ind_t_cnr:
                    c.cnr = t.tcnr];
```

In the combination phase, a reference relation, refrel, is built to evaluate the AND operator:

```
refrel :=     [<c1.cref,c2.tref> OF
   EACH c1 IN sl_csoph, EACH c2 IN ij_c_t
               : c1.cref = c2.cref];
```

4.0 QUERY OPTIMIZATION STRATEGIES

In the previous section, we outlined an algorithm that can evaluate general first-order relational expressions. In this section, we discuss some approaches we have followed in the PASCAL/R query evaluation system to improve the efficiency of query evaluation.

Basically, there are two lines of attack. Firstly, one can transform a query so that its evaluation avoids repeated access to identical data and keeps intermediate data structures small. Secondly, one can optimize the representation of intermediate structures and operations critical for the efficiency of the evaluation process. In this paper, we comment on query transformation; specific representation techniques are discussed in [8].

Generally speaking, PASCAL/R's query transformations have the effect of shifting work load from the combination phase to the collection phase in order to decrease combinatorial growth inherent in the combination of intermediate results. Transformation and evaluation of relational expressions are guided by the following strategies:

1. Form subexpressions so that as many computations on a database relation as possible can be done "in parallel" without access to other database relations.

2. Determine nested subexpressions that can be evaluated in one step.

3. Decrease the cardinality of range relations.

4. Recognize special cases and provide specific evaluation techniques for them.

4.1 Parallel Evaluation Of Subexpressions

In section 3.3, each relation is accessed as many times as variables ranging over it occur in (different) join terms. Strategy 1 builds subexpressions that can be evaluated in parallel; thus, each range relation is read no more than once [11].

EXAMPLE 4.1:

In EXAMPLE 3.2, the subexpression

```
(c.clevel <= sophomore) AND (c.cnr = t.tcnr)
```

of the query example is transformed into the subexpression

```
    [<cs.cref,ct.tref> OF
 EACH cs IN [<@c> OF EACH c IN courses:
                     c.clevel <= sophomore],
 EACH ct IN [<@c,s.tref> OF
         EACH s IN [<t.tcnr,@t> OF
                EACH t IN timetable: true],
         EACH c IN courses
            : c.cnr = s.tcnr]
    : cs.cref = ct.cref]
```

The two subexpressions defining cs and ct require access to the relation, courses. If the subexpression ranging over timetable is already

evaluated before processing of courses starts, both subexpressions can be evaluated in parallel while reading the relation one-element-at-a-time [15].

4.2 Evaluation Of Nested Subexpressions

By our first strategy, subexpressions referring to the same database relation are evaluated in parallel. Evaluation of a nested subexpression, however, is not started, before all of its subexpressions have been evaluated, i.e. in the combination phase.

Strategy 2 determines those nested subexpressions that can be evaluated in one step. Candidates are conjunctions of join terms over the same variable. If such a conjunction contains both monadic and dyadic terms, the monadic terms can be used to restrict the indirect joins for the dyadic terms; consequently, single lists need not be constructed. If there are no dyadic terms over the variable, one single list represents the relation elements which satisfy all monadic join terms in the conjunction.

EXAMPLE 4.2:

The subexpression

```
(c.clevel <= sophomore) AND (c.cnr = t.tcnr)
```

can be evaluated in one step while reading the relation, courses:

```
refrel := [];
FOR EACH c IN courses: true DO
    IF c.clevel <= sophomore
    THEN refrel_31 :+ [<@c,t.tref> OF
                        EACH t IN ind_t_cnr:
                             t.tcnr = c.cnr];
```

Note, that this technique also allows two indirect joins to restrict each other; details are given in [7,8]. In many cases, however, the evaluation of a conjunction of dyadic join terms has to be partially deferred to the combination phase.

Finally, the parallel evaluation of join terms is demonstrated for EXAMPLE 2.2 by a sequence of element-oriented PASCAL/R statements.

EXAMPLE 4.3:

```
FOR EACH t IN timetable:  true DO
BEGIN
   ind_t_cnr :+ [<t.tcnr,@t>];
   ind_t_enr :+ [<t.tenr,@t>]
END;
FOR EACH c IN courses:  true DO
IF (c.clevel <= sophomore)
THEN FOR EACH t IN ind_t_cnr:
                t.tcnr = c.cnr DO
     ij_c_t :+ [<@c,t.tref>];
FOR EACH p IN papers:  true DO
BEGIN
   IF p.pyear <> 1977
   THEN sl_p77 :+ [<@p>];
   ind_p_enr :+ [<p.penr,@p>]
END;
FOR EACH e IN employees: true DO
BEGIN
   IF e.estatus = professor
   THEN sl_prof :+ [<@e>];
   IF e.estatus = professor
   THEN FOR EACH t IN ind_t_enr:
                   t.tenr = e.enr DO
        ij_e_t :+ [<@e,t.tref>];
   IF e.estatus = professor
   THEN FOR EACH p IN ind_p_enr:
                   p.penr <> e.enr DO
        ij_e_p :+ [<@e,p.pref>]
END;
```

4.3 Extended Range Expressions

The cardinality of range relations has a very strong impact on the time and storage consumption of query evaluation. Therefore, our third strategy looks for query transformations that replace database range relations given by the user by relational expressions on these database relations.

EXAMPLE 4.4:

The reduction of intermediate structures shown in EXAMPLE 4.2 can be achieved in a totally different way. The range expression for c is extended from the database relation courses to the relational expression

[EACH c IN courses: c.clevel <= sophomore].

By this transformation, the subexpression displayed in EXAMPLE 4.1 simplifies to

```
   [<@c,s.tref> OF
 EACH c IN [EACH c IN courses:
                 c.clevel <= sophomore],
 EACH s IN [<t.tcnr, @t> OF
            EACH t IN timetable: true]
    : c.cnr = s.tcnr]
```

Although, for this example, the effect of strategy 3 is the same as that of strategy 2, in general the extension of range expressions is more efficient for query optimization, because it works on a query as a whole and takes into account the quantifiers. The distinction between the two strategies has, to our knowledge, not been recognized in the literature.

Given a standard form selection expression, the PASCAL/R compiler can find the appropriate monadic expression, say S(rec), to extend the range expression of some variable, rec. To do so, it uses the equivalences

```
SOME rec IN rel (S(rec) AND WFF) =
SOME rec IN [EACH r IN rel:  S(r)] (WFF)
```

for existentially quantified variables and

```
ALL rec IN rel (NOT (S(rec)) OR WFF) =
ALL rec IN [EACH r IN rel:  S(r)] (WFF)
```

for universally quantified variables. Free variables are handled as if existentially quantified.

The current system version supports only conjunctions of join terms as range expression extensions. The use of the more general conjunctive normal form is expected to improve further the efficiency of the system by reducing the number of conjunctions in the disjunctive form of the remaining matrix.

The subsequent example demonstrates the advantages of extended range expressions over strategy 2.

EXAMPLE 4.5:

Provided all range relations are non-empty, our sample query can be transformed to

```
enames :=
[<e.ename> OF
 EACH e IN [EACH e IN employees:
                  e.estatus = professor]:
  ALL p IN [EACH p IN papers: p.pyear = 1977]
  SOME c IN [EACH c IN courses:
                  c.clevel <= sophomore]
  SOME t IN timetable
  ((p.penr <> e.enr)
   OR
   (t.tenr = e.enr) AND (t.tcnr = c.cnr))]
```

A comparison with EXAMPLE 4.3 shows that most profit may be gained in the case of a universally quantified variable. There is one conjunction less to be evaluated and the size of indirect joins is reduced considerably. In addition, the evaluation of the employees relation is more efficient, as e.estatus=professor is tested only once for each element.

4.4 Specific Techniques For Special Cases

Bottlenecks arise in the combination phase when the intermediate results from the collection phase are combined into larger reference relations - in most cases just to be reduced again to a comparatively small set of references to qualified relation elements. Strategy 4 breaks up the strict phase structure by moving quantifiers into the matrix and evaluating them in the collection phase.

Consider a selection expression with f free variables (v1, ... , vf) and n-f quantified variables (v(f+1), ... , vn). The matrix in disjunctive form is a disjunction of conjunctions

"... OR c (v1, ... , vf, ... , vn) OR ..."

where c consists of AND-connected join terms.

As the quantifiers must be evaluated from right to left, we can restrict our attention to vn. In which cases can the quantifier of vn be evaluated already in the collection phase?

1. Let vn be existentially quantified. Each conjunction can be evaluated separately and, by Lemma 1, we can shift the quantifier to the right of those terms in which vn does not occur. The quantified sub-wff can be evaluated separately.

2. If vn is universally quantified, splitting is possible only by Lemma 1, if vn occurs in no more than one conjunction. This conjunction can be split arbitrarily, if the range relation of vn is non-empty.

EXAMPLE 4.6:

In EXAMPLE 2.2, p occurs in two conjunctions; no immediate quantifier evaluation seems possible. On the other hand, in EXAMPLE 4.5, p occurs in only one conjunction, because extended range expressions are applied. Quantifier evaluation can be performed in the collection phase.

The possibility of splitting a formula is of interest only if there is a way to evaluate the quantified sub-formula more efficiently than by the standard algorithm.

Here, we restrict ourselves to those cases where there is only one additional variable (say vm) in the quantified sub-formula, i.e. it consists only of dyadic join terms over vn and vm in addition to monadic terms over vn. This can often be achieved by swapping quantifiers. Quantifiers may be swapped, if they are equal, or by application of the various forms of Lemma 1.

The technique works as follows. When vnrel is read, instead of a complete index only its value list is generated. Afterwards, when vmrel is read, the quantifier of vn can be evaluated, because it can immediately be decided for each element of vmrel whether it corresponds to SOME respectively ALL elements of vnrel or not. The further handling is similar to that for monadic join terms. Note that a similar approach was chosen to resolve chained queries in SQL [18]; however, the user has to decide upon chaining.

For queries without universal quantifiers where arbitrary quantifier swapping is allowed, thorough theoretical results recently developed in a distributed system environment [2,3] show under which circumstances the system can fully resolve a query in the collection phase.

The advantages of quantifier evaluation in the collection phase can be increased by application of the following considerations:

If the relational operator of the join term connecting vm and vn is < or <= (symmetric considerations hold for > or >=), only one component value of vnrel must be stored. If vn is existentially quantified, this is the maximum value, otherwise it is the minimum value of the value list. The

reason is that "less than SOME vn component value" is equivalent to "less than the maximum" and "less than ALL values" means "less than the minimum".

Also, if the relational operator = occurs combined with the quantifier ALL of vn, or the operator <> combined with quantifier SOME, at most one value need to be stored. Either there is only one component value in vnrel, or the subexpression can at once be said to be false in the ALL case and true in the SOME case, because a value in virel cannot be equal to two different values in vnrel. This argument assumes that there is a value in virel, i.e., virel is non-empty. If virel is empty, the result of the subexpression is known anyway.

EXAMPLE 4.7:

We apply strategy 4 to EXAMPLE 4.5 where the quantifier sequence of t and c is changed. So we have the query:

```
enames :=
[<e.ename> OF
 EACH e IN [EACH e IN employees:
                  e.estatus = professor]:
  ALL p IN [EACH p IN papers: p.pyear = 1977]
     (p.penr <> e.enr)
  OR
  SOME t IN timetable
     ((t.tenr = e.enr) AND
      SOME c IN [EACH c IN courses:
                       c.clevel <= sophomore]
          (c.cnr = t.tcnr))]
```

The evaluation can be sketched by the following sequence of statements:

```
cset:= [<c.cnr> OF
        EACH c IN [EACH c IN courses:
                     c.clevel <= sophomore]
           : true];
tset:= [<t.tenr> OF EACH t IN timetable:
         SOME c in cset (c.cnr = t.tcnr)];
pset:= [<p.penr> OF
        EACH p IN [EACH p IN papers:
                         p.pyear = 1977]
           : true];
enames:=
[<e.ename> OF
 EACH e IN [EACH e IN employees:
                  e.estatus = professor]:
  SOME t IN tset (t.tenr = e.enr)
  OR
  ALL p in pset (p.penr <> e.enr)];
```

5.0 SUMMARY AND CONCLUSIONS

Logic and evaluation of relation-valued expressions with quantifiers and various comparison operators were analyzed in the framework of the integrated database language PASCAL/R. The language construct "reference" was introduced to allow a high-level language implementation.

In the area of query optimization, we believe our contributions to be the following. Firstly, the widely used technique of pre-evaluating monadic join terms within a conjunction of join terms was extended: Parallel evaluation of join terms while reading a relation allows dyadic join terms to restrict each other and the extension of range expressions is a more global way to exploit monadic join terms. Secondly, it was demonstrated, how "semi-join" techniques [2,3] can be interpreted from a general first-order predicate calculus point of view and extended to the case of universal quantifiers.

The paper concentrated on logic-based transformations. Ongoing research tries to integrate them with optimal use of permanent access paths and to develop further approaches to improve the phase structure of the algorithm.

ACKNOWLEDGMENTS

Manuel Mall gave important input to the definition of selected variables. We would also like to thank Juergen Koch, Ted Stohr, and one of the referees for helpful comments which improved the English presentation of the paper.

REFERENCES

1. M.M.Astrahan et al.: System R: Relational Approach to Database Management, ToDS 1 (1976), 97-137

2. P.A.Bernstein, D.-M.W.Chiu: Using Semi-Joins to Solve Relational Expressions, JACM 28 (1981), 25-40

3. P.A.Bernstein, N.Goodman, E.Wong, C.L.Reeve, J.B.Rothnie: Query Processing in a System for Distributed Databases (SDD-1), ToDS 6 (1981), 602-625

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

5. E.F.Codd: Relational Completeness of Data Base Sublanguages, Courant Computer Science Symposium, New York 1971, 65-98

6. L.R.Gotlieb: Computing Joins of Relations, Proc. ACM-SIGMOD Conf., San Jose 1975

7. M.Jarke: Entwurf und Implementation von Algorithmen zur Behandlung von Anfragen an relationale Datenbanken, unpubl. diploma thesis, Hamburg 1978 (in German)

8. M.Jarke, J.W.Schmidt: Evaluation of First-Order Relational Expressions, IFI-HH-B-78/81, Hamburg 1981

9. W.Kim: A New Way to Compute the Product and Join of Relations, Proc. ACM-SIGMOD Conf., Santa Monica 1980

10. W.Kim: Query Optimization for Relational Databases, IBM RJ3081, San Jose 1981

11. F.P.Palermo: A Database Search Problem, 4th Computer and Information Science Symposium, Miami Beach 1972, 67-101

12. W.V.O.Quine: Mathematical Logic, 5th ed., Cambridge/Mass. 1965

13. A.Schmidt: Ueber deduktive Theorien mit mehreren Sorten von Grunddingen, Mathematische Annalen 115 (1938), 485-506 (in German)

14. J.W.Schmidt: Some High-Level Language Constructs for Data of Type Relation, ToDS 2 (1977), 247-261

15. J.W.Schmidt: Parallel Processing of Relations: A Single-Assignment Approach, Proc. 5th VLDB, Rio de Janeiro 1979

16. J.W.Schmidt: Generalized Selection Mechanisms for Relations (forthcoming)

17. J.W.Schmidt, M.Mall: PASCAL/R Report, IFI-HH-B-72/81, Hamburg 1981

18. P.G.Selinger et al.: Access Path Selection in a Relational Database Managament System, Proc. ACM-SIGMOD Conf., Boston 1979

19. E.Wong, K.Youssefi: Decomposition - A Strategy for Query Processing, ToDS 1 (1976), 223-241

FIGURE 1: DECLARATION OF SAMPLE DATABASE

```
TYPE statustype = (student, technician, assistant, professor);
     nametype   = PACKED ARRAY [1..10] OF char;
     titletype  = PACKED ARRAY [1..40] OF char;
     roomtype   = PACKED ARRAY [1..5] OF char;
     yeartype   = 1900..1999;
     timetype   = 08000900..18002000;
     daytype    = (monday,tuesday,wednesday,thursday,friday);
     leveltype  = (freshman,sophomore,junior,senior);
     enumbertype= 1..99;
     cnumbertype= 1..99;

VAR  employees  : RELATION <enr> OF
                  RECORD
                     enr          : enumbertype;
                     ename        : nametype;
                     estatus      : statustype
                  END;

     papers     : RELATION <ptitle,penr> OF
                  RECORD
                     penr         : enumbertype;
                     pyear        : yeartype;
                     ptitle       : titletype
                  END;

     courses    : RELATION <cnr> OF
                  RECORD
                     cnr          : cnumbertype;
                     clevel       : leveltype;
                     ctitle       : titletype
                  END;

     timetable  : RELATION <tenr,tcnr,tday> OF
                  RECORD
                     tenr         : enumbertype;
                     tcnr         : cnumbertype;
                     tday         : daytype;
                     ttime        : timetype;
                     troom        : roomtype
                  END;
```

FIGURE 2: DECLARATION OF AUXILIARY STRUCTURES

```
(* single lists *)
VAR sl_prof : RELATION <eref> OF RECORD eref : @employees END;
    sl_p77  : RELATION <pref> OF RECORD pref : @papers END;
    sl_csoph: RELATION <cref> OF RECORD cref : @courses END;

(* indirect joins *)
    ij_c_t      : RELATION <cref,tref> OF
                  RECORD
                     cref : @courses;
                     tref : @timetable
                  END;
    ij_e_t      : RELATION <eref,tref> OF
                  RECORD
                     eref : @employees;
                     tref : @timetable
                  END;
    ij_e_p      : RELATION <eref,pref> OF
                  RECORD
                     eref : @employees;
                     pref : @papers
                  END;

(* indexes *)
    ind_t_enr   : RELATION <tenr,tref> OF
                  RECORD
                     tenr : enumbertype;
                     tref : @timetable
                  END;
    ind_t_cnr   : RELATION <tcnr,tref> OF
                  RECORD
                     tcnr : cnumbertype;
                     tref : @timetable
                  END;
    ind_p_enr   : RELATION <penr,pref> OF
                  RECORD
                     penr : enumbertype;
                     pref : @papers
                  END;
```