A SURVEY OF QUERY OPTIMIZATION IN CENTRALIZED DATABASE SYSTEMS

Matthias Jarke
New York University

and

Juergen Koch
Universitat Hamburg

November 1982

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

Working Paper Series

CRIS #44

GBA #82-73(CR)

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

ABSTRACT

Efficient ways to process unanticipated queries are a crucial prerequisite for the success of generalized database management systems. A wide variety of approaches for improving the performance of query evaluation algorithms have been proposed: logic-based and semantic transformations, fast implementations of basic operations, and combinatorial or heuristic algorithms for generating and choosing among alternative access plans. This paper surveys these approaches in the framework of a general query evaluation procedure using the relational calculus representation of queries. The focus is on centralized database systems; some relationships to other system types are studied.

Acknowledgment

This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG) under grant no. SCHM 450/2-1.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

TABLE OF CONTENTS

INTRODUCTION

1. THE QUERY OPTIMIZATION PROBLEM

1.1 Queries
1.2 Optimization Objectives
1.3 Top-Down Approach to Query Optimization

2. QUERY REPRESENTATION

2.1 The Relational Calculus
2.2 The Relational Algebra
2.3 Query Graphs
2.4 Tableaux

3. QUERY TRANSFORMATION

3.1 Standardization
3.2 Simplification
3.3 Amelioration

4. QUERY EVALUATION

4.1 One-Variable Expressions
4.2 Two-Variable Expressions
4.3 Multi-Variable Expressions

5. ACCESS PLANS

5.1 Generation of Access Plans
5.2 Selection of Access Plans
5.3 Support for Multiple Queries

SUMMARY AND CONCLUSIONS

REFERENCES

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

INTRODUCTION

Database management systems (DBMS) have become a standard tool for shielding the computer user from details of secondary storage management. This is supposed to improve the productivity of application programmers and to facilitate access of computer-naive end users.

There have been two major research areas in database systems. One is the analysis of data models into which the real world can be mapped and on which user interfaces for different user types can be built. Such conceptual models include the hierarchical, the network, the relational, and a number of semantic-oriented models and have been reviewed in a large number of books and surveys [CHAM76], [TAYL76], [TSIC76], [KIM79].

A second area of interest is safe and efficient implementation. Data have become a central resource of most organizations. Each implementation meant for production use must take this into account by guaranteeing safety of the data in the cases of concurrent access [BERN81e], recovery [VERH78], and reorganization [SOCK79].

On the other hand, a main criticism of many early DBMS's has been their lack of efficiency in handling the general operations they offer, especially the content-based access to data by queries.

Query optimization tries to solve this problem. In doing so, it has to integrate a large number of techniques and strategies ranging from logical transformations of queries to the optimization of access paths and the storage of data on the file system level.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Traditionally, all these approaches use different languages to describe their solutions. This is probably one of the reasons that there has not been a comprehensive survey of query optimization techniques so far.

The goal of this paper is to discuss query optimization techniques in the common framework of the relational calculus representation of queries. This has been shown to be technically equivalent to a relational algebra representation [CODD72], [KLUG82a], and extendible to the implementation of network DBMS [DAYA82]. For the sake of a reasonable size, the paper concentrates on query optimization in centralized DBMS. This means, that the following related areas will not be treated in detail:

User optimization: The overall cost of an information system is composed of the DBMS cost and the costs of user efforts to work with the system. The two areas interface in the functionality and usability of the query language [VASS82], [JARK82b], especially the response time of the system. If one assumes given functional capabilities of the query language and a response time minimization objective of the query optimization system, query optimization can be handled as a separately tractable subproblem of user optimization.

File Structures: A query optimization algorithm has to choose among a variety of existing access paths to resolve a query. The internal details of implementing such access paths, however, and the derivation of the related cost factors, are beyond the scope of this paper.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Distributed DBMS: The physical distribution of data in a DBMS has been the subject of much recent research in concurrency control and query optimization. Distributed query processing is related to centralized processing in the sense that it uses the techniques presented here for local pre-processing but the specific communications-related approaches will not be discussed in detail.

The paper is organized as follows. Section 1 gives a global framework for query optimization. A top-down approach integrates the different levels discussed in subsequent sections.

Section 2 compares four techniques for representing queries in terms of their suitability for optimization. Section 3 mostly relies on one of these techniques, the relational calculus, for presenting logic-based transformation methods of query optimization. The emerging area of semantic query optimization is addressed briefly.

After being transformed, a query must be mapped into a sequence of operations that finally return the requested result. Section 4 analyzes the implementation of such operations on a given low-level system of stored data and access paths. Finally, section 5 presents optimization procedures to integrate these operations into a globally optimal access plan.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

1. THE QUERY OPTIMIZATION PROBLEM

This section states the objectives of query optimization and presents a general procedure in order to structure the solution process.

1.1 Queries

A query is a language expression that describes data to be retrieved from a database. In the context of query optimization, it is often assumed that queries are expressed in a content-based (and often set-oriented) manner which gives sufficient choice among alternative evaluation procedures.

Queries are used in several settings. The most obvious application are direct requests by end users who need information about structure or content of the database. If there are only standard requests, queries can be optimized manually by programming the associated search procedures and offering the user menu selection or similar techniques. However, an automatic query optimization system becomes necessary, if ad hoc queries are asked using a general purpose query language.

A second application is the use of queries in transactions which change the stored data in a way that is based on their current value ("give all professors a 10% salary increase").

Finally, query-like expressions are used internally in a DBMS, for example, to check integrity constraints.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

1.2 Optimization Objectives

The economic principle requires that any optimization either tries to maximize the output for a given amount of resources, or to minimize the resource usage for a given output. Query optimization tries to minimize the response time for a given query language and mix of query types (if known) in a given system environment.

In order to allow for a fair comparison of efficiency, the functional capabilities of the systems to be compared must be similar. The requirement of "relational completeness" coined by [CODD72] (compare section 2.1, below) has become a quasi-standard. Therefore, the techniques surveyed in this paper are presented as contributions to implementing queries in a relationally complete language with minimal evaluation cost (i.e., response time).

The total cost to be minimized is the sum of

Communication Cost: The cost of transmitting data from the site where they are stored to the sites where computations are performed and results are presented. These costs are composed of costs for the communication line which are usually related to the time the line is open, and of costs for the delay in processing caused by transmission. The latter - more important for query optimization - is often assumed to be proportional to the amount of data transmitted.

Secondary Storage Access Cost: The cost of (or time for) loading data pages from secondary storage into main memory.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Storage Cost: The cost of occupying secondary storage and memory buffers over time. Storage costs are relevant only, if storage becomes a system bottleneck and is variable between queries. The latter is usually not true.

Computation Cost: The cost for (or time of) using the CPU.

The structure of query optimization algorithms is strongly influenced by the trade-off among these cost components.

In long-range distributed DBMS, communication delay dominates the costs while the other factors are only relevant for local sub-optimization.

In centralized systems, the stress is on minimizing the time for secondary storage accesses although, for complex queries, the CPU costs may be quite high [GOTL75].

Finally, in locally distributed DBMS, all factors have similar weights resulting in very complex cost functions and optimization procedures.

This paper focuses on query optimization in centralized DBMS. This can be justified as follows. Centralized query optimization appears as a subproblem in distributed systems. In addition, centralized query optimization is a problem in its own right in many large main-frame databases and, more recently, in personal microcomputer DBMS.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The consequence of concentrating on centralized query optimization is that communication costs do not enter the decision because communication requirements (with users and storage devices) are independent of the evaluation strategy. For the optimization of single queries, secondary storage costs are irrelevant as well; therefore, they will be considered only where one deals with the simultaneous optimization of several queries.

There remain the costs of secondary storage accesses (usually measured by the number of page accesses) and of CPU usage (often measured by the number of comparisons to be performed). A number of common ideas underly most techniques to reduce these factors: they try to avoid double work, to use standardized parts, to look ahead to avoid unnecessary operations, to choose elementary operations, and to sequence them in an optimal way. The following example may indicate how this can be achieved.

Consider a relational database with the schema (key attributes are underlined)

employees (<u>enr</u>, ename, status, city)
papers (<u>enr</u>, <u>title</u>, year)
departments (<u>dname</u>, city, street-address)
courses (<u>cnr</u>, cname, abstract)
lectures (<u>cnr</u>, <u>dname</u>, <u>enr</u>, daytime)

The database describes employees offering certain lectures to departments of a geographically distributed organization. Employees are characterized by their status, the city where they live, and the papers they have written.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Assume that someone wants to know which departments in New York offer courses on database management. There are many possible strategies to solve this query four of which will be compared, below. Assume first, that all relations are physically sorted by ascending key values and that no special access paths such as indexes exist.

Strategy 1

1. Merge courses and lectures.
2. Sort the result by dnames.
3. Merge the result with departments. Concurrently,
4. select the combinations with city = 'New York' and cname = 'database management', and
5. keep only the dname column.

This strategy is obviously very costly because it generates an intermediate result which is roughly the product of the participating relations. Therefore, one would like to reduce the size of the relations to be sorted and merged.

Strategy 2

1. Select the departments with city = 'New York' and keep only the dname column.
2. Select the courses with cname = 'database management' and keep only the cnr column.
3. Merge the cnr list with the lectures relation and keep only the dname column.
4. Sort the dname column gained in step 3.
5. Merge the dname columns of steps 1 and 4.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Strategy 2 reduces the relations to be processed by selecting only the necessary elements and projecting onto those columns that are needed for further processing. However, the procedure can be further improved by using a pipelining technique for the quasi parallel execution of steps 2 and 3 of strategy 2.

Strategy 3

1. Merge courses with lectures, keeping only the dname field of combinations with cname = 'database management'.
2. Sort the dname list generated.
3. Merge the list with the departments relation, keeping only dnames with city = 'New York'.

Step 1 avoids the creation and subsequent reading of an intermediate result (the cnr list of strategy 2). This is advantageous because the two relations, courses and lectures, have the same sort order. Otherwise, the reduction in sorting costs induced by the selection and projection in step 2 of strategy 2 would offset the relatively small cost of the cnr list.

Assume now that two indexes exist in the database, i.e., for the cnr and the dname attributes of the lectures relation. An index can be viewed as a binary relation, associating an attribute value with references to the corresponding relation elements. However, indexes are implemented in a way that allows for fast retrieval by the value attribute. Assume further that the sort order of references is the same as that of the underlying relations (note, that this does not hold in a general paged environment).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Strategy 4

1. Select the dname for departments with city = 'New York'. Concurrently, for each such dname, retrieve the corresponding index elements (giving references to the relation, lectures) and store them in a binary relation (partial dname index).
2. Sort the partial dname index by ascending lectures reference. Note, that the above assumption implies that the references are also sorted by ascending cnr and dname.
3. Select the cnr for courses on 'database management'. Concurrently, for each such cnr, retrieve the matching lectures references from the cnr index and merge them with the partial dname index keeping only the dnames.
4. Sort the resulting dname list to remove duplicates if unique names are desired.

If reference sort order does not follow element sort order, a partial cnr index similar to the partial dname index must be constructed and sorted before merging. In neither case, however, access to the relation, lectures, is required.

The effect of query optimization can be quite impressive. Assume the following physical storage data for the above example.

There are 100 departments 5 of which are located in New York. A physical block can take 5 department records or 50 dname values.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

There are 500 courses 20 of which are on database management. Physical block size is 10 records or 50 cnames.

There are 2000 lectures of which 300 present database management courses, 100 are held New York departments, and 20 (from 3 departments) satisfy both conditions. Physical block size is 10 records or 50 references.

Assume further that sorting time is N*log(2)N where N is the file size in blocks, and that there is a buffer of one block for each relation.

Under these assumptions, the costs of the four strategies (number of secondary storage accesses) are approximately

Strategy 1: 3100

Strategy 2: 450

Strategy 3: 450 (actually 2 less than strategy 2)

Strategy 4: 150

Thus, a reduction by a factor of approximately 20 has been achieved. For larger databases, more complex queries, and more sophisticated techniques much higher reductions are often possible.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

1.3 Top-Down Approach to Query Optimization

Query optimization research in the literature can be divided in two classes which would contrast as being bottom-up and top-down.

On one hand, researchers soon detected that the overall query optimization problem is very complex. Therefore, theoretical work started with special cases such as the optimal implementation of important operations or evaluation strategies for certain sub-classes of queries. Subsequent research tried to compose larger building blocks from these early results.

On the other hand, a need for working systems triggered the development of full-scale evaluation procedures [PALE72], [ASTR75], [WONG76]. They stressed the generality of solutions and handled query optimization in a uniform and heuristic manner. As this often did not achieve competitive system efficiency, the current trend seems to be to incorporate more knowledge about special case optimization into the general procedures. At the same time, the general algorithms themselves are augmented by combinatorial cost-minimization procedures for choosing among strategies.

This paper follows the second approach. The following general evaluation procedure serves as a framework for the specific techniques developed in query optimization research.

Step 1: Find an internal query representation into which user queries can easily be mapped and which leaves the system all degrees of freedom to optimize the evaluation.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Step 2: Apply logical transformations to the query representation that (1) standardize the query in order to simplify the next transformation steps, (2) simplify the query to avoid double work, and (3) ameliorate the query to streamline the evaluation and to allow special case procedures to be applied.

Step 3: Map the transformed query into alternative groups and sequences of elementary operations for which a good implementation and the associated costs are known. The result of this step is a set of candidate "access plans".

Step 4: Compute the overall cost for each access plan, choose the cheapest one, and execute it.

The first two steps of this procedure are to a large degree data-independent and thus can be handled mostly at compile time. The quality of the steps 3 and 4, that is, the richness of the access plans generated and the optimality of the choice algorithm, heavily depend on knowledge about data values in the database.

This has two consequences. First, these steps can only be done at runtime, which means that the possible gain in efficiency must be traded off against the cost of the optimization itself. Second, a meta-database (e.g., an augmented data dictionary) must maintain general information about the database structure and statistical information about the database contents. As in many similar operational research problems (e.g., inventory control), the costs of this additional information system must be compared with the value of its information.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

2. QUERY REPRESENTATION

Queries can be represented in a number of forms. In the context of query optimization an appropriate query representation form has to fulfill the following requirements:

- it should be powerful enough to express a large class of queries;
- it should provide a well-defined base for query transformation.

In the following, we shall present four different query representation forms each of which has been used in a number of approaches to query optimization.

2.1 The Relational Calculus

The relational calculus as introduced in [CODD71], [CODD72] is a notation for defining the result of a query through the description of its properties. The representation of a query in relational calculus consists of two parts, i.e., the target list and the selection expression.

The selection expression specifies the contents of the relation resulting from the query by means of a first-order predicate, i.e., a generalized Boolean expression possibly containing existential and/or universal quantifiers. The target list defines the free variables occurring in the predicate, and specifies the structure of the resulting relation. Example 2.1 demonstrates the relational calculus representation using the syntax of the database programming language Pascal/R [SCHM77].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Example 2.1:

Names of professors who published some paper in 1981.

```
[<e.name> OF
 EACH e IN employees: e.status = professor
                          AND
                      SOME p IN papers
                         (e.enr = p.enr AND p.year = 1981)]
```

In the target list, i.e., in the subexpression preceding the colon, the range of the (free) variable e is restricted to elements of the relation employees. The relation employees is therefore called the range relation of e. The term '<e.name>' indicates that only the names of employees are considered in the result.

The predicate following the colon defines constraints on the free variable. The first constraint is a monadic term restricting the free variable to those employees who have the status professor. This constraint is AND-connected with a dyadic term, relating employees to papers, and another monadic one, further restricting the result to those employees who published some paper in 1981. The comparison operators usually allowed in terms are =, #, <, >, <=, and >=.

As opposed to the one-sorted predicate calculus, the relational calculus allows variables to be bound to different sorts (range relations, e.g. variable e is bound to employees and variable p is bound to papers). The consequences of the many-sortedness of the relational calculus with respect to query transformation are discussed in section 3.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

In addition to the logical operator AND, the operators OR and NOT can also be used in predicates. Relational calculus predicates are completely defined by the following recursive rules:

1. A (monadic or dyadic) term is a predicate.
2. Let A be a predicate, rec a tuple variable, and rel a relation. Then
   (i) SOME rec IN rel (A)
   (ii) ALL rec IN rel (A)
   are predicates.
3. Let A and B be predicates. Then
   (i) NOT (A) (negation)
   (ii) A AND B (conjunction)
   (iii) A OR B (disjunction)
   are predicates.
4. No other formulae are predicates.

In [CODD72] the relational calculus has been introduced as a yardstick of expressive power. A representation form is said to be _relationally complete_ if it allows the definition of any query result definable by a relational calculus expression.

Clearly, relational completeness has to be considered as a minimum requirement with respect to expressive power. An often-cited example for a conceptually simple query which goes beyond relational completeness is "find the names of employees reporting to manager Smith at any level", provided a hierarchy of employees is modeled in a single relation (e.g., via a name and manager attribute). In general, the computation of the transitive

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

closure requires a calculus of higher order [PIRO79].

Furthermore, queries in today applications often contain aggregations which cannot be expressed in pure relational calculus. However, the extension of relational calculus by aggregate functions is rather straightforward [KLUG82b].

2.2 The Relational Algebra

The relational algebra as defined in [CODD72] is a collection of operators on relations. These operators fall in two classes, i.e., traditional set operators such as Cartesian product, union, intersection, and difference, and special relational algebra operators such as restriction, projection, join, and division. The special operators will be defined, below, by relating them to equivalent relational calculus expressions.

The _restriction_ operator applied to a relation R constructs a horizontal subset of its elements according to a simple predicate.

Rest(R,pred) = [EACH r IN R: pred]

Restriction predicates are usually monadic terms, intra-relational dyadic terms (both variables are bound to the same relation), or conjunctions thereof.

The _projection_ operator serves for the construction of a vertical subset of a relation R by selecting a set A of specified attributes and eliminating duplicate tuples within these attributes.

Proj(R,A) = [<A> OF EACH r IN R: true]

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The join operator permits two relations R and S to be combined into a single relation whose attributes are the union of the attributes of the relations R and S.

```
Join(R, A op B, S) = [EACH r IN R, EACH s IN S: r.A op s.B]
```

The comparison operators op allowed in joins are the same as those in dyadic terms of the relational calculus. If op is the equality operator '=', the "natural" join omits either A or B in the result.

The division operator provides an algebraic counterpart to the universal quantifier. It is defined as follows:

```
Divi(R, A by B, S) = [<compl(A)> OF EACH r IN R:
                                  ALL s IN S SOME t IN R
                                  (t.compl(A) = r.compl(A) AND
                                  t.A = s.B)]
```

where compl(A) is the complement of A in the attribute set of R.

The definition indicates, that division is a rather complex operation which can make the understanding of a query a difficult job.

Example 2.2 represents the query of example 2.1 in relational algebra.

Example 2.2:

Names of professors who published some paper in 1981.

```
Proj(Rest(Join(employees,
               enr=enr,
               Rest(papers,year=1981)),
          status=professor),
     name)
```

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

As opposed to a relational calculus expression which describes the relation resulting from the query by means of its properties, a relational algebra expression defines an algorithm for the construction of the resulting relation. Therefore, a calculus expression appears to be a better starting point for query optimization since it provides an optimizer only with the basic properties of the query which might get hidden in a particular sequence of algebra operators.

With respect to relational completeness, however, the relational algebra is at least as powerful as the relational calculus. In [CODD72] it has been shown, that any relational calculus expression can be translated into an equivalent algebra expression. An analogous result for algebra and calculus expressions extended by aggregate functions has been proven in [KLUG82a].

2.3 Query Graphs

Graphs have been used for the visual representation of structured objects in a number of areas. Two well-known examples are the use of syntax trees in compiler construction and the use of AND/OR graphs in artificial intelligence applications.

In the context of query optimization, graphs are used for the representation of queries or query evaluation strategies. Two classes of graphs can be distinguished: object graphs and operator graphs.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Nodes in <u>object graphs</u> represent objects such as (relation) variables and constants. Edges describe predicates that these objects are to fulfill [PALE72], [WONG79]. Object graphs contain the properties of the query result and are therefore closely related to the relational calculus.

<u>Operator graphs</u> describe an operator-controlled data flow by representing operators as nodes which are connected by edges indicating the direction of data movement. In [SMIT75], [YAO79] operator graphs have been used for the representation of algebra expressions. Figures 2.1 and 2.2 give one example each for an object graph and an operator graph.

Query graphs have many attractive properties. The visual presentation of a query often leads to an easier understanding of its structural characteristics. In addition, graph theory offers a number of results useful for the automatic analysis of graphs (e.g., discovery of cycles, tree property). And finally, a main advantage of query graphs is that they can be easily augmented with additional information. The augmentation of graphs with details of the physical data organization of a database is discussed in subsection 5.1.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

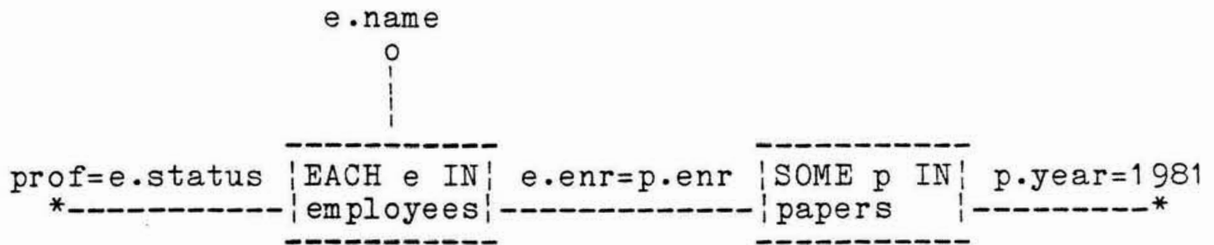

Figure 2.1 An object graph representing the example query.

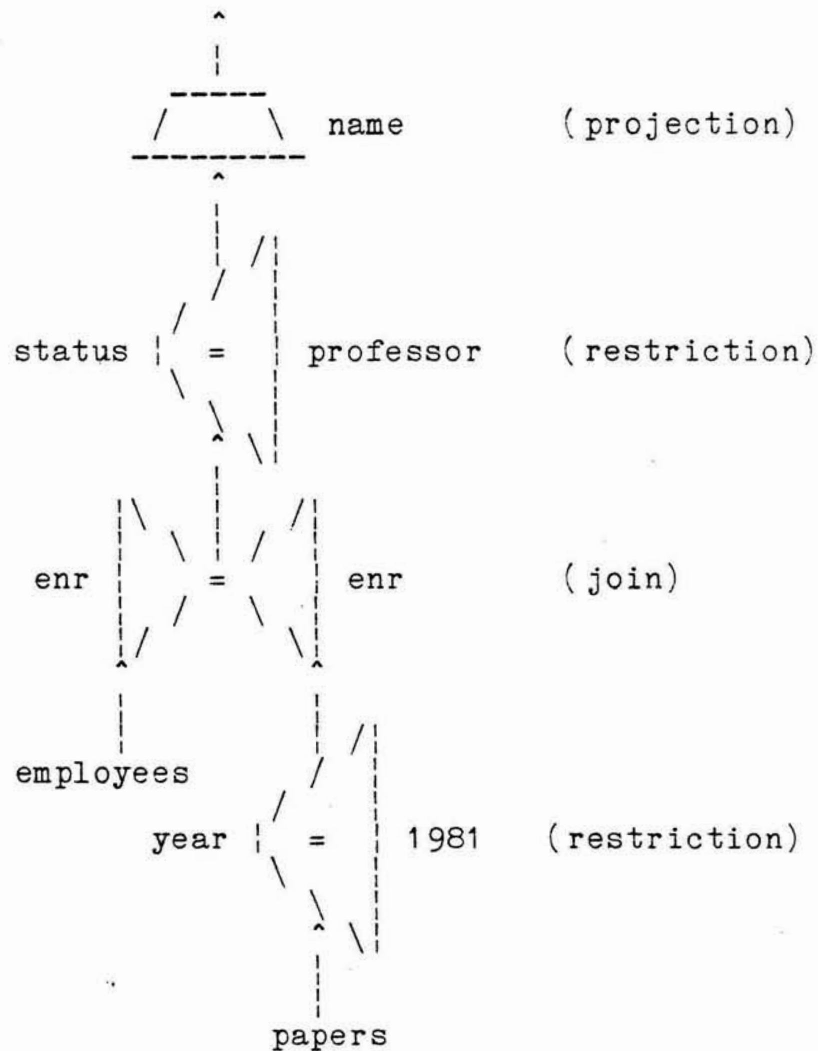

Figure 2.2: An operator graph representing the example query.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

2.4 Tableaux

Tableaux as defined in [AHO79a-c] are a tabular notation for a subset of relational calculus queries characterized by containing no universal quantifiers and only AND-connected terms. Such queries are frequently referred to as conjunctive queries [ROSE80],[CHAN77].

Tableaux are specialized matrices the columns of which correspond to the attributes of the underlying database. The first row of the matrix, called the summary, serves the same purpose as the target list of a relational calculus expression. The other rows describe the predicate.

The symbols appearing in a tableau are distinguished variables (representing free variables), nondistinguished variables (representing existentially quantified variables), constants, blanks, and tags (indicating the range relation).

The construction of a tableau T representing the query of example 2.1 starting with tableaux for single relations and proceeding with combining these tableaux to new tableaux for larger and larger subexpressions is illustrated in figure 2.3. Distinguished variables are denoted by a's, nondistinguished ones by b's.

Expressions containing disjunctions (unions) and negations (differences) can be represented by sets of tableaux [SAGI80]. Therefore, tableau sets can be considered a relationally complete representation form. The specific value of tableaux with respect to query optimization will be discussed in section 3.2.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

```
                  status    name     enr     year
                 ------------------------------------
T(employees) =  |  a1        a2       a3              |
                 ------------------------------------
                |  a1        a2       a3              |  employees
                 ------------------------------------

                 ------------------------------------
T(papers) =     |                     a3       a4     |
                 ------------------------------------
                |                     a3       a4     |  papers
                 ------------------------------------

T( Rest(papers,year=1981) ) =
                 ------------------------------------
                |                     a3      1981    |
                 ------------------------------------
                |                     a3      1981    |  papers
                 ------------------------------------

T( Join(employees,
        enr=enr,
        Rest(papers,year=1981) ) =
                 ------------------------------------
                |  a1        a2       a3      1981    |
                 ------------------------------------
                |  a1        a2       a3              |  employees
                |                     a3      1981    |  papers
                 ------------------------------------

T( Rest(Join(employees,
             enr=enr,
             Rest(papers,year=1981),
        status=professor) ) =
                 ------------------------------------
                | professor  a2       a3      1981    |
                 ------------------------------------
                | professor  a2       a3              |  employees
                |                     a3      1981    |  papers
                 ------------------------------------

T( Proj(Rest(Join(employees,
                  enr=enr,
                  Rest(papers,year=1981),
             status=professor),
        name) ) =
                 ------------------------------------
                |            a2                       |
                 ------------------------------------
                | professor  a2       b3              |  employees
                |                     b3      1981    |  papers
                 ------------------------------------
```

Figure 2.2: Stepwise construction of a tableau representing the query of example 2.1

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

3. QUERY TRANSFORMATION

Queries cannot only be expressed in a number of different representation forms, but also, for each expression representing some query there may exist a number of semantically equivalent expressions of the same form. The transformation of an expression into an equivalent one by means of well-defined rules is the subject of this section. We distinguish three goals of query transformation:

- the construction of a standardized starting point for query optimization (standardization);
- the elimination of redundancy (simplification);
- the construction of ameliorated expressions with respect to evaluation performance (amelioration).

3.1 Standardization

Several approaches to query optimization define a standardized starting point through a normalized version of the underlying query representation form [PALE72], [WONG76], [JARK81], [KIM82]. In the following, we shall present two normal forms for the relational calculus together with the rules to be obeyed by the normalization procedure.

A relational calculus representation of a query is said to be in prenex normal form if its selection expression is of the form

SOME/ALL r1 IN rel1 .. SOME/ALL rn IN reln (M)

where M is a quantifier-free predicate.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

M is called the <u>matrix</u> and can also be standardized. A matrix consisting of a disjunction of conjunctions (of terms Aij) such as

(A11 AND .. AND A1n) OR .. OR (Am1 AND .. AND Amn)

is said to be in <u>disjunctive normal form</u>, and a matrix consisting of a conjunction of disjunctions such as

(A11 OR .. OR A1n) AND .. AND (Am1 OR .. OR Amn)

is in <u>conjunctive normal form</u>.

The prenex normal form combined with the normal forms for the matrix yields two normal forms for relational calculus expressions: <u>disjunctive prenex normal form</u> (DPNF) and <u>conjunctive prenex normal form</u> (CPNF). The use of DPNF is motivated by the goal to keep intermediate results small during the evaluation of a query, since in many cases disjunctions can be evaluated independently of each other [BERN81a], [JARK81]. The CPNF has proven useful for the reduction of queries into serializable components [WONG76] and for data-dependent amelioration (e.g., testing the most restrictive disjunction first).

The transformation of an arbitrary relational calculus expression into prenex normal form is a matter of moving quantifiers over terms (from right to left). Quantifier movement is governed by the transformation rules of table 3.1.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

```
Q1: A AND SOME r IN rel (B(r))
      <==>
    SOME r IN rel (A AND B(r))

Q2: A OR SOME r IN rel (B(r))
      <==>
 a) SOME r IN rel (A OR B(r))  | rel # []
 b) A                          | rel = []

Q3: A AND ALL r IN rel (B(r))
      <==>
 a) ALL r in rel (A AND B(r))  | rel # []
 b) A                          | rel = []

Q4: A OR ALL r IN rel (B(r))
      <==>
    ALL r IN rel (A OR B(r))

Q5: SOME r1 IN rel1 SOME r2 IN rel2 (A(r1,r2))
      <==>
    SOME r2 IN rel2 SOME r1 IN rel1 (A(r1,r2))

Q6: ALL r1 IN rel1 ALL r2 IN rel2 (A(r1,r2))
      <==>
    ALL r2 IN rel2 ALL r1 IN rel1 (A(r1,r2))

Q7: SOME r IN rel (A(r) OR B(r))
      <==>
    SOME r IN rel (A(r)) OR SOME r IN rel (B(r))

Q8: ALL r IN rel (A(r) AND B(r))
      <==>
    ALL r IN rel (A(r)) AND ALL r in rel (B(r))

Q9: NOT ALL r IN rel (A(r))
      <==>
    SOME r IN rel (NOT(A(r)))

Q10:NOT SOME r IN rel (A(r))
      <==>
    ALL r IN rel (NOT(A(r)))
```

Table 3.1: Transformation rules for quantified expressions

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

M1: Commutative Laws

a) A OR B <==> B OR A

b) A AND B <==> B AND A

M2: Associative Laws

a) (A OR B) OR C <==> A OR (B OR C)

b) (A AND B) AND C <==> A AND (B AND C)

M3: Distributive Laws

a) A OR (B AND C) <==> (A OR B) AND (A OR C)

b) A AND (B OR C) <==> (A AND B) OR (A AND C)

M4: Idempotency Laws

a) A OR A <==> A

b) A AND A <==> A

c) A OR NOT(A) <==> TRUE

d) A AND NOT(A) <==> FALSE

e) A OR FALSE <==> A

f) A AND TRUE <==> A

g) A OR TRUE <==> TRUE

h) A AND FALSE <==> FALSE

i) NOT (NOT (A)) <==> A

M5: De Morgan's Laws

a) NOT (A AND B) <==> NOT (A) OR NOT (B)

b) NOT (A OR B) <==> NOT (A) AND NOT (B)

Table 3.2: Transformation rules for the matrix

The distinction of the cases between empty and non-empty range relations in rules Q2 and Q3 of table 3.1 is due to the many-sortedness of the relational calculus [JARK82a].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

A relational calculus expression can be transformed into an equivalent expression of a one-sorted calculus by introducing a range definition such as (r IN rel) as another type of atomic predicate:

```
01: SOME r IN rel (pred)
       <==>
    SOME r ((r IN rel) AND pred)
02: ALL r IN rel (pred)
       <==>
    ALL r ((r IN rel) ==> pred)
```

The application of rules Q2a and Q3a when moving a quantifier over a term would therefore yield a wrong result in the case of an empty range relation. It follows, that an automatic normalization of an arbitrary relational calculus expression must preserve information about the original range definition of variables, so that runtime modifications according to rules Q2b and Q3b can be performed when necessary.

Normalization of the matrix is rather straightforward and can be achieved by using DeMorgan's laws, the distributive laws, and the law of double negation (see table 3.2).

3.2 Simplification

We have already seen that there might be several semantically equivalent expressions representing one and the same query. One source of origin for differences between any two equivalent expressions is their degree of redundancy [HALL76], [STRO79]. A straightforward evaluation of a redundant expression would lead to the execution of a set of operations some of which are superfluous.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Therefore, query optimization aims at the elimination of redundancy by means of transforming a redundant expression into an equivalent non-redundant one.

A redundant expression can be simplified by applying the transformation rules M4a to M4i considering idempotency (see table 3.2). The application of these rules is complicated by the fact that idempotency can occur at any level in the expression due to the presence of common subexpressions, i.e., subexpressions occuring more than once in the expression representing the query. Thus, in order to simplify an expression like

```
[EACH e IN employees:
      e.name = 'Smith'
        OR
      (e.status = assistant OR e.status = professor)
        AND
      NOT(e.status = professor OR e.status = assistant)]
```

to

```
[EACH e IN employees: e.name = 'Smith']
```

by means of rules M1a and M4g, the subexpressions

```
(e.status = assistant OR e.status = professor)
```

and

```
(e.status = professor OR e.status = assistant)
```

must first be recognized as being equivalent. Algorithms for the recognition of common subexpressions are given in [HALL74a].

The recognition of common subexpressions and the application of idempotency rules have to be performed concurrently rather than sequentially, since the simplification of an expression by means of idempotency rules may yield further common subexpressions which in turn are subject to simplification.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Expressions that are bound to empty relations can also be simplified. Transformations rules for their simplification are given in table 3.3. Note, that these rules can only be applied at runtime.

```
E1: [EACH r in []: pred]                      <==> []
E2: [<r.A1,..,r.An> OF EACH r IN []: pred]    <==> []
E3: SOME r IN [] (pred)                       <==> FALSE
E4: ALL r IN [] (pred)                        <==> TRUE
```

Table 3.3: Transformation rules for expressions with empty relations

Terms serve as atomic predicates in the relational calculus (section 2.1). However, the matrix can be simplified if the semantic of the comparison operators is taken into account explicitly. One important application is the so-called constant propagation which uses transitivity laws such as

```
r.A op s.B AND s.B = const  ==> r.A op const
```

to reduce the number of dyadic terms in a query. Algorithms that minimize the number of rows in tableaux as introduced in section 2.4 systematically exploit such simplification rules for conjunctive queries [AHO79a]. Since the number of rows in a tableau is one more than the number of joins (dyadic join terms) in the expression, the minimization of the number of rows corresponds to the elimination of redundant joins.

[SAGI80] extends the tableau techniques to cover the simplification of expressions containing disjunctions. The generalization to all expressions of a relationally complete language, however, is still an open problem [SAGI81].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

3.3 Amelioration

Query simplification does not necessarily produce a unique expression in the sense that there may exist other non-redundant expressions which are semantically equivalent to the one generated by some simplification technique. The evaluation of expressions corresponding to one and the same query may differ substantially with respect to performance parameters such as the size of intermediate results, the number of relation elements accessed, etc. In the following, we present a set of query transformation heuristics which, when applied to expressions, yield ameliorated expressions with respect to evaluation performance.

The simplest transformations considered in this section are the combination of a sequence of projections into a single projection and the combination of a sequence of restrictions into a single restriction [SMIT75], [HALL76]. The corresponding transformation rules are:

```
A1:  Proj(..Proj(Proj(R,A1),A2),..,An)
        <==>
     Proj(R,An)

A2:  Rest(..Rest(Rest(R:pred1):pred2):..:predn)
        <==>
     Rest(R:pred1 AND pred2 AND .. AND predn)
```

Combining intrarelational operations has two advantages. First, repetitive reading of the same relation is avoided, and second, existing access paths may be used for the combined operation and not only for the first operation in the sequence.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Minimization of the size of intermediate results to be constructed, stored, and retrieved is the goal of a number of ameliorating transformations. One important heuristic moves selective operations such as restriction and projection over constructive operations such as join and Cartesian product in order to perform the selective ones as early as possible [SMIT75]. In the context of relational calculus, the consideration of a certain evaluation sequence can be represented by a nested expression. The evaluation of a nested expression starts with the evaluation of the innermost nesting, followed by its surrounding nesting and so on until the outermost nesting is reached. A nested expression implying the early evaluation of monadic terms (restrictions) is given in exmple 3.1.

<u>Example 3.1</u>: A nested expression equivalent to the expression in example 2.1.

```
[<e.name> OF
 EACH e IN [EACH e IN employees: e.status = professor]:
   SOME p IN [EACH p IN papers: p.year = 1981]
     (e.enr = p.enr)]
```

The early evaluation of selective operations is a special case of the so-called <u>query detachment</u> as introduced in [WONG76]. There, a subexpression that overlaps with the rest of the expression on a single variable is detached and forms an inner nesting. This is done recursively at any nesting level until the expression cannot be further reduced. Example 3.2 demonstrates the detachment of a subexpression in a complex expression.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

<u>Example 3.2</u>: Departments offering lectures that are held by professors who live in the same city where the department is located and who have published some paper in 1981.

The corresponding expression:

```
[EACH d IN departments:
   SOME l IN lectures
     SOME e in employees
       (e.status = professor
         and
       d.dname=l.dname AND l.enr=e.enr AND e.city=d.city
         AND
       SOME p IN papers
         (p.year = 1981 AND p.enr = e.enr))]
```

An equivalent expression produced by query detachment:

```
[EACH d IN departments:
   SOME l IN lectures
     SOME e IN [EACH e IN
                 [EACH e IN employees: e.status = professor]:
                 SOME p IN [EACH p IN papers: p.year = 1981]
                   (e.enr = p.enr)]
       (d.dname=l.dname AND l.enr=e.enr AND e.city=d.city)]
```

An object graph representing the query is shown in Figure 3.1.

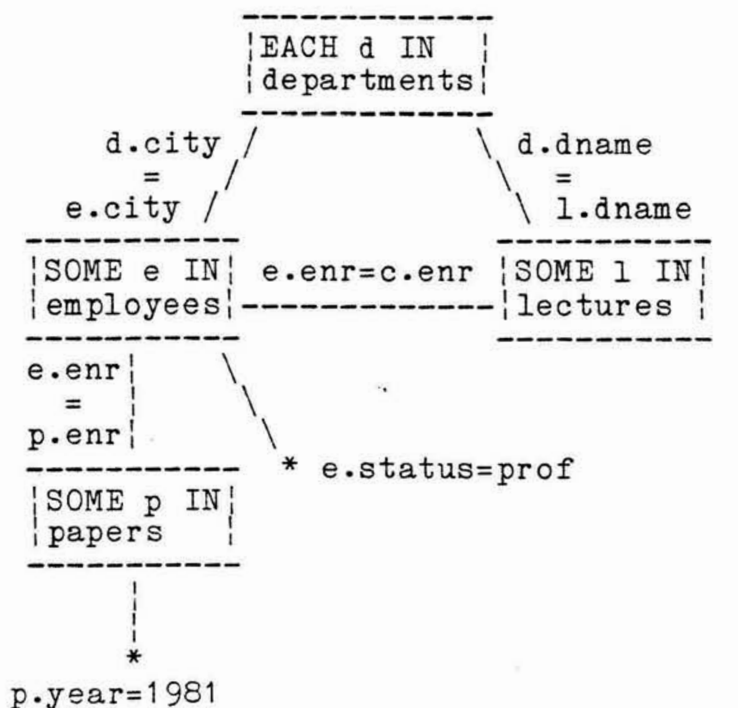

Figure 3.1: Object graph for example 3.2

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Note that the resulting nested expression is irreducible since it cannot be separated into two subexpressions overlapping on a single variable. In other words, the nested expression contains a cycle. This is also obvious from figure 3.1.

The importance of the distinction between cyclic and acyclic (tree-like) expressions for query processing will be further discussed in subsection 4.3. At this point, we shall mention only that there are cycles which can be further reduced, for example those that are introduced by transitivity. In [BERN81b], [BERN81d], algorithms for the detection of such benign cycles are described.

The concept of *extended range expressions* [JARK82a] provides a generalization of query detachment in that it also considers expressions containing universal quantification. Database relations defining the range of a relation variable are replaced by calculus expressions according to the following transformation rules:

```
A3: [EACH r IN rel: pred1 AND pred2]
      <==>
    [EACH r IN [EACH r IN rel: pred1]: pred2]

A4: SOME r IN rel (pred1 AND pred2)
      <==>
    SOME r IN [EACH r IN rel: pred1] (pred2)

A5: ALL r IN rel: (NOT(pred1) OR pred2)
      <==>
    ALL r IN [EACH r IN rel: pred1] (pred2)
```

Note that transformation rule A5 for universally quantified variables is especially profitable since through the reduction of the number of conjunctions in the outer nesting, the intermediate results can be expected to be considerably smaller in size.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The ameliorating transformations presented so far use information from three sources: general transformation rules and heuristics guiding their usage, knowledge about the relational data structures, and the query itself. Two other knowledge bases have not been considered: integrity constraints that complement the structural data definition in many database systems, and the data itself.

Integrity constraints are predicates that must be true for each element of a certain relation or for each combination of elements of a certain group of relations (referential integrity). Thus, they can be added to the selection expression of any query without changing its truth value. There are a few approaches making use of this observation under the labels of knowledge-based [HAMM80] or <u>semantic query processing</u> [KING81].

Assume for example, that one integrity constraint says: "we hire only professors who have at least one paper per year". In this case, the evaluation of example 2.1 (asking for professors with papers in 1981) becomes trivial, and the evaluation of example 3.2 is substantially simplified.

Adding an integrity constraint to a selection expression can also change the structure of the query to make it more tractable. Consider the constraint: "we hire only local professors". In this case, the term 'd.city=e.city' in example 3.2 can be omitted. The remaining query does no longer contain a cycle.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The success of semantic query processing depends largely on the development of efficient heuristics for choosing among the many transformations made possible by adding any combination of integrity constraints to the query. QUIST [KING81] uses artificial intelligence type rules to make this decision for a special class of relational databases.

[YAO79] points out that there are cases where the optimal transformation is data-dependent. The heuristics presented above may not always be optimal, especially when certain access paths are supported by physical data structures. One consequence of such data dependence is that, in addition to the query compiler, also the runtime support must be equiped with query transformation facilities.

Furthermore, if heuristics do not yield satisfactory results simultaneous optimization of the physical and the logical level becomes necessary. Before turning to such integrated approaches, however, the physical evaluation of query components has to be described.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

4. QUERY EVALUATION

This section presents methods for the evaluation of query components of varying complexity such as one-variable expressions, two-variable expressions, and multi-variable expressions. The individual approaches can be viewed as the building blocks of a general query evaluation system. Their associated costs and ranges of applicability constitute the input to the last stage of the query optimization process which generates the optimal access plan.

4.1 One-Variable Expressions

One-variable expressions describe conditions for the selection of elements from a single relation. A naive approach to their evaluation would be to read every element of the relation and test if it satisfies each term of the expression. Since this approach is very costly, especially in the presence of large relations and complex expressions, various techniques have been used to improve its efficiency. These techniques aim at reducing the number of element accesses, and at reducing the number of tests applied to an accessed element.

The number of element accesses can be reduced by employing data structures that provide access paths other than the exhaustive sequential access. One possibility is to keep the relation sorted with respect to one or more attributes so that it can be accessed in ascending or descending order. This has proven useful for the evaluation of range expressions, i.e., expressions that define an interval of attribute values.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Direct and ordered access is also provided by indexes. An index is a relation which associates attribute values with references to relation elements, usually called tuple identifiers (TIDs). We distinguish <u>one-dimensional indexes</u> supporting access via a single relation attribute, and <u>multi-dimensional indexes</u> supporting access via a set of attributes. One-dimensional indexes are usually implemented by ISAM [IBM66] or B-tree [BAYE72] structures. An overview of multi-dimensional index structures is given in [BENT79].

The number of tests applied to an accessed relation element during expression evaluation can be reduced by means of runtime transformations of the expression. The optimization of a special class of expressions, <u>Boolean expressions</u>, has since long been a research topic in compiler construction [GRIE71]. Boolean expressions, i.e., quantifier-free AND/OR connected terms are an integral part of a number of control structures in high-level programming languages. The overall idea for the code optimization of Boolean expressions is to generate code that skips over the evaluation of expression components no longer relevant to the value of the expression as a whole. For example in the statement

```
IF A AND B THEN
  statement_1
ELSE
  statement_2
END
```

the evaluation of term B is superfluous and the ELSE-branch can be executed right away in case term A has already been evaluated to FALSE.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The same idea applied to the evaluation of one-variable expressions in query languages [GUDE73], [LIU76] can be interpreted as query simplification at runtime.

Changing the order in which individual expression components are evaluated is another approach to improve evaluation efficiency. Several algorithms are known to lead to optimal evaluation sequences in certain situations with [HANA77], and without [BREI75] considering a priori probabilities for attribute values.

4.2 Two-Variable Expressions

Two-variable expressions describe conditions for the combination of elements from two relations. In general, two-variable expressions are composed of monadic terms, restricting single variables independently of each other, and dyadic terms, establishing the link between both variables. In this subsection we shall first describe the basic methods for the evaluation of a single dyadic term corresponding to the join operator defined in subsection 2.2 and then discuss strategies for the evaluation of arbitrary two-variable expressions.

Approaches to the implementation of the join operation can be classified into those, where the order in which relation elements are accessed is relevant, and into those, where it is not. The basic method independent of the order of element access is the so-called nested iteration method [PECH76], [SELI79]. In this method, every pair of relation elements is accessed and concatenated if the join condition is satisfied.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

A sketch of the algorithm follows:

```
FOR i := 1 TO N1 DO
  read i-th element of R1;
  FOR j := 1 TO N2 DO
    read j-th element of R2;
    form the join according to the join condition;
  END;
END;
```

Let N1 (N2) be the number of elements of the relation read in the outer (inner) loop, then N1 + N1*N2 secondary storage accesses are required to evaluate the dyadic term assuming that each element access needs one secondary storage access.

The nested iteration method can be augmented by the use of an <u>index</u> on the join attribute(s) of R2. Instead of scanning R2 sequentially for each element of R1, the matching R2 elements are retrieved directly [KLUG82b]. Thus, only N1 + N1*N2*j12 accesses are required where j12 is a join selectivity factor describing the reduction of the Cartesian product R1*R2 by the join condition.

The <u>nested block method</u> [KIM80] adapts the nested iteration method to a paged-memory environment. The method assumes a main memory buffer which holds one or more pages of both relations. Each page contains a set of relation elements.

The algorithm itself is basically identical to the one of the nested iteration method except that memory pages are read instead of single relation elements. The number of secondary storage accesses needed to form the join is reduced to P1 + (P1/B1)*P2 where P1 (P2)

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

is the number of pages occupied by the outer (inner) relation and B1 is the number of pages of the outer relation held in the main memory buffer. The formula makes clear that it is always preferable to read the smaller relation in the outer loop (that is, to make P1 < P2). Note that only P1 + P2 accesses are necessary, if one of the relations can be kept entirely in the main memory buffer.

The merge method [BLAS77], [SELI79] is based on a certain order in which relation elements are accessed. Both relations are scanned in ascending or descending order of join attribute values and merged according to the join condition. Approximately N1 + N2 + S2 + S2 secondary storage accesses are required where S1 and S2 denote the number of secondary storage accesses necessary to sort the relations. In case the relations are already sorted or indexes are available, the merge method appears to be the most efficient approach to evaluate a dyadic term [MERR81b].

Methods for the evaluation of arbitrary two-variable expressions are combined of strategies for one-variable expressions and algorithms for the computation of dyadic terms. They differ in the way they make use of and/or temporarily create access paths such as indexes, links, and sorting, and in the order in which the terms are processed. One such method applied to the evaluation of the query in example 2.1 is illustrated in the operator graph of figure 4.1.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

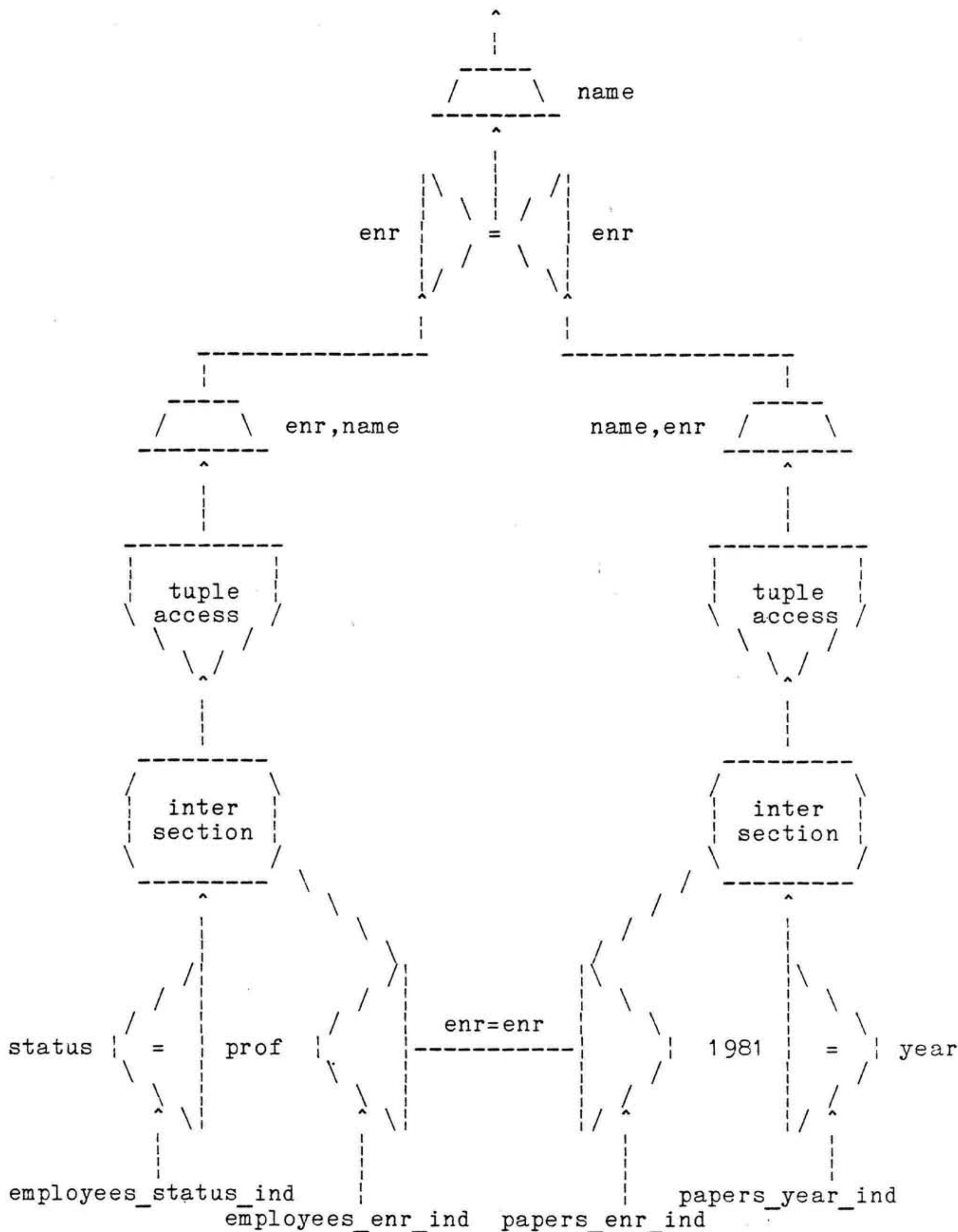

Figure 4.1: Operator graph illustrating the evaluation of example query 2.1. The existence of various indexes is assumed.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The method makes extensive use of indexes. Tuple identifiers resulting from the processing of monadic terms and those that satisfy the join condition are intersected and then used to access the relation elements. These elements are projected onto attributes appearing in the dyadic term and in the target list. The projected elements are concatenated and projected on the target attribute.

In [BLAS76] and [YAO79] various other algorithms are presented and systematically compared with respect to their efficiency. Their results demonstrate that often no a priori best algorithm exists so that one has to rely either on heuristics or on an expensive cost comparison of many alternatives for each query.

4.3 Multi-Variable Expressions

Strategies for the evaluation of multi-variable expressions, i.e., expressions containing at least two variables, are the largest building blocks for a general query processing system. There are two basic approaches, which will be referred to as parallel processing and stepwise reduction.

The parallel processing of query components serves to avoid repeated access to the same data, and creation and subsequent reading of temporary results. Repeated access to the same data can be avoided by simultaneous evaluation of multiple query components. In [PALE72], all monadic terms associated with some variable are completely, and the dyadic terms partially processed. Existing AND conections among terms can also be exploited in parallel to reduce the size of intermediate results [JARK81].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

A similar approach on a higher level is taken in [KLUG82b] where aggregate functions and complex subqueries are computed in parallel. Scheduling strategies for the parallel processing of query components are discussed in [SCHM79].

The pipelining of operations that can work on partial output of preceding operations is another technique that exploits parallelism [SMIT75], [YAO79]. For example, restriction and projection can be pipelined so that only a relatively small buffer for data exchange is needed instead of the creation and subsequent reading of a temporary relation.

Aspects of simultaneous evaluation and pipelining are combined in the so-called feedback method [ROTH74], [CLAU80]. The idea is to use partial results of a join operation in order to restrict its input. The degree to which this can be done depends on the quantification of variables occuring in the join term. For example, consider the expression

[EACH r IN R: ALL s IN S (r.A op s.B)].

Assume that the join term is evaluated by nested iteration. While testing some element, r, it is found that r.A op s.B is false for a certain s with s.B = c1. Because of the universal quantification of s, r is rejected, and an elimination filter can be added:

[EACH r IN R: NOT (r.A op c1) AND ALL s IN S (r.A op s.B)]

because the same s would fail all r that do not satisfy the first term. On the other hand, if r with r.A = c2 passes the test, a true-filter can be added [CLAU80]:

[EACH r IN R: r.A op c2 OR NOT (r.A op c1) AND ...].

Both filters can be updated subsequently to sharpen the constraints.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The second basic approach to the evaluation of multi-variable expressions will be motivated by means of the following example.

Example 4.1:

Figure 4.2 shows an object graph representing the expression

```
[<d.dname> OF
  EACH d IN departments:
    SOME e IN employees (e.status=professor
                         AND e.city=d.city)
      AND
    SOME l IN lectures (l.daytime>8pm
                        AND l.dname=d.dname)]
```

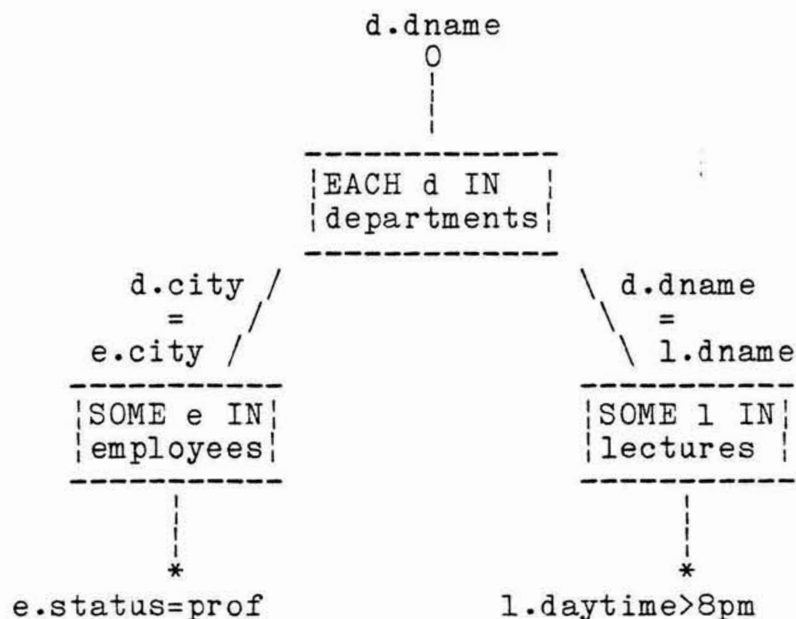

Figure 4.2: Object graph for example 4.1

Expressions like the one in example 4.1 are called <u>tree expressions</u> [GOOD82], [SHMU81] since their associated query graph is a tree. A simple approach to evaluating such an expression would be to form the join of the three relations, restrict the intermediate result according to monadic terms and finally project it onto the attributes appearing in the target list. As shown already in the

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

introductory example (section 1.2, strategy 1), this approach performs very poorly. This is even more problematic in a distributed environment where each relation resides on a different site. The reasons are that entire relations are transmitted from site to site, and that the relation at the target site is temporarily expanded through the formation of the join, although the final result will only be a horizontal and vertical subset of it.

In [BERN81d] the <u>stepwise reduction</u> of tree expressions (with free and existentially quantified variables) has been introduced which often outperforms the simple approach above, in a decentralized as well as in a centralized setting. The method is based on a modified join operation, i.e., the so-called semijoin.

The <u>semijoin</u> of a relation R by a relation S equals the join of these relations projected back onto the attributes of relation R [BERN81a]. The advantage of the semijoin is, that its evaluation only requires the transmission of a value-list of the join attributes, instead of an entire relation, since only 'half of a join' is to be formed.

The evaluation of a tree expression by means of stepwise reduction proceeds as follows: starting from the leaves of the query tree representing the expression, one semijoin per edge is executed in breadth-first leaf-to-root order. Thus, a tree expression containing n variables is completely processed by n-1 semijoins if every variable except the one corresponding to the root of the tree is existentially quantified. An additional semijoin procedure in reversed order is required in case all variables are free.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Strategies for the evaluation of tree expressions containing both existential and universal quantifiers must take into account the order in which these quantifiers appear in the expression.

Stepwise reduction is only possible when the processing of the edges of the query tree (breadth-first leaf-to-root) corresponds to the order of the quantifiers in the expression (right to left).

Example 4.2:

Consider the query tree of figure 4.3 which represents the expression

```
[<d.dname> OF
  EACH d IN departments:
    ALL p IN papers
      SOME e IN employees
        (p.enr=e.enr AND e.city=d.city)
      AND
    SOME l IN lectures (l.dname=d.dname)]
```

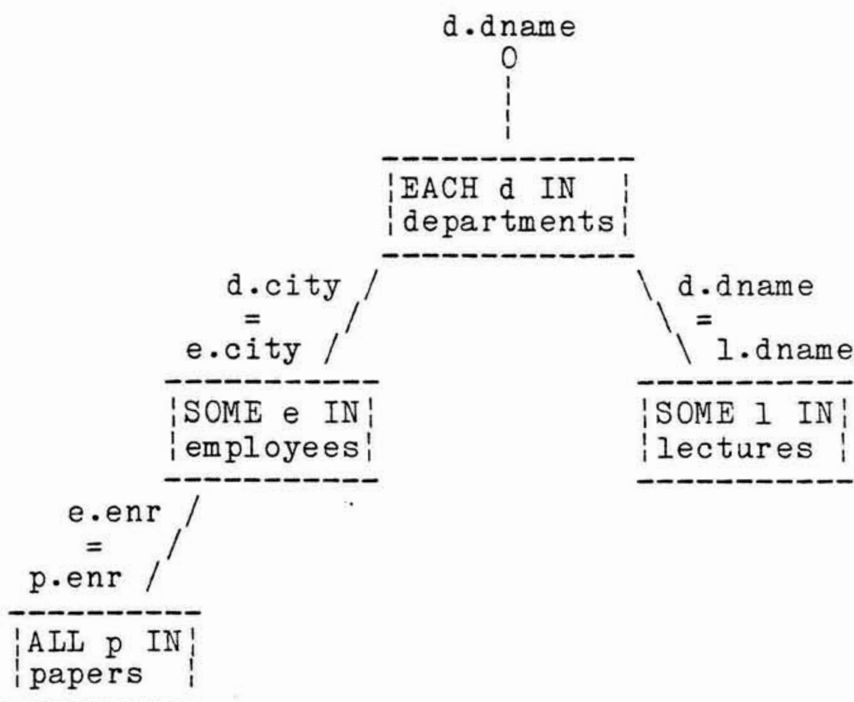

Figure 4.2: Object graph for example 4.2

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Processing the tree in breadth-first leaf-to-root order would yield the value of the expression

```
[<d.dname> OF
  EACH d IN departments:
    SOME e IN employees
      ALL p IN papers
        (p.enr=e.enr AND e.city=d.city)]
      AND
    SOME l IN lectures (l.dname=d.dname)
```

which is not equivalent to the original expression.

The position of an existential and a universal quantifier cannot be interchanged without changing the meaning of the expression, except in the cases of rules Q1 through Q4 of table 3.1. The problem does not occur in expressions containing only one sort of quantifiers since their positions can be arbitrarily interchanged according to transformation rules Q5 and Q6.

Cyclic expressions are the complement of tree expressions with respect to the entire set of expressions. Although there are some benign exceptions [BERN81c], cyclic expressions in general can not be stepwise reduced by means of semijoins [GOOD81].

Example 4.3:

Consider the query: "names of departments that offer lectures after 8pm given by professors who live in the city where the department is located." The corresponding relational calculus expression and a query graph are shown in figure 4.5.

There is no sequence of semijoins corresponding to edges of the query graph of example 4.3 that produces the correct result (the empty relation) if the database is in the state shown in figure 4.4. The reason is that the semijoin technique only considers one edge at a time, and thus looses restrictive conditions introduced through the feedback effect of the cycle.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

departments	dname	city	street-address
	d1	ci1	s1
	d2	ci2	s2

employees	enr	ename	status	city
	e1	en1	st1	ci1
	e2	en2	st2	ci2

lectures	cnr	enr	dname	daytime
	c1	e1	d2	da1
	c2	e2	d1	da2

Figure 4.2: Some possible database state.

```
[<d.dname> OF
  EACH d IN departments:
    SOME e IN employees
      (e.status=professor
        AND
      SOME l IN lectures
        (l.daytime>8pm
          AND
        d.dname=l.dname AND l.enr=e.enr AND e.city=d.city))]
```

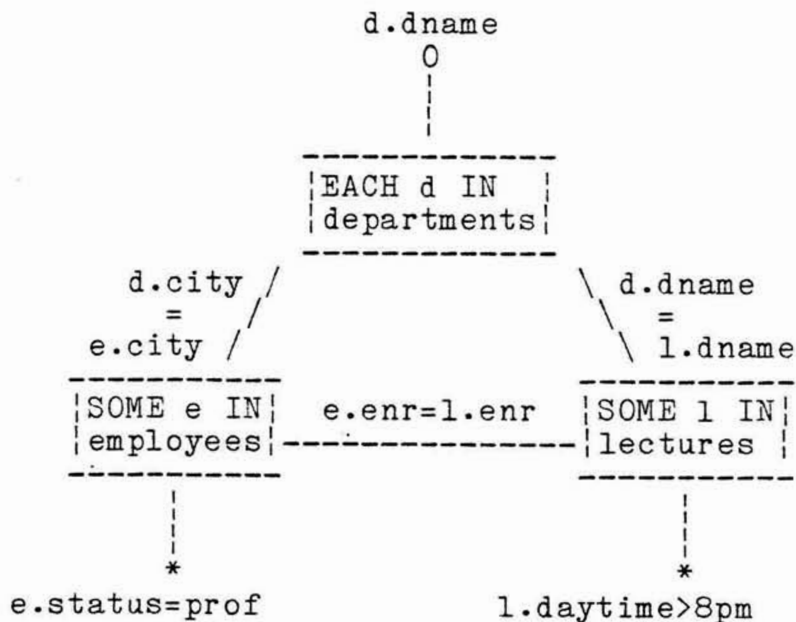

Figure 4.5: Calculus expression and object graph for example 4.3

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

In [KAMB82] a proposal is made that is supposed to generalize the applicability of the semijoin technique to cyclic expressions. The overall idea is to transform the cyclic query graph into a tree by adding appropriate terms to some edges of the graph. Figure 4.6 demonstrates the technique applied to the cyclic expression of example 4.3.

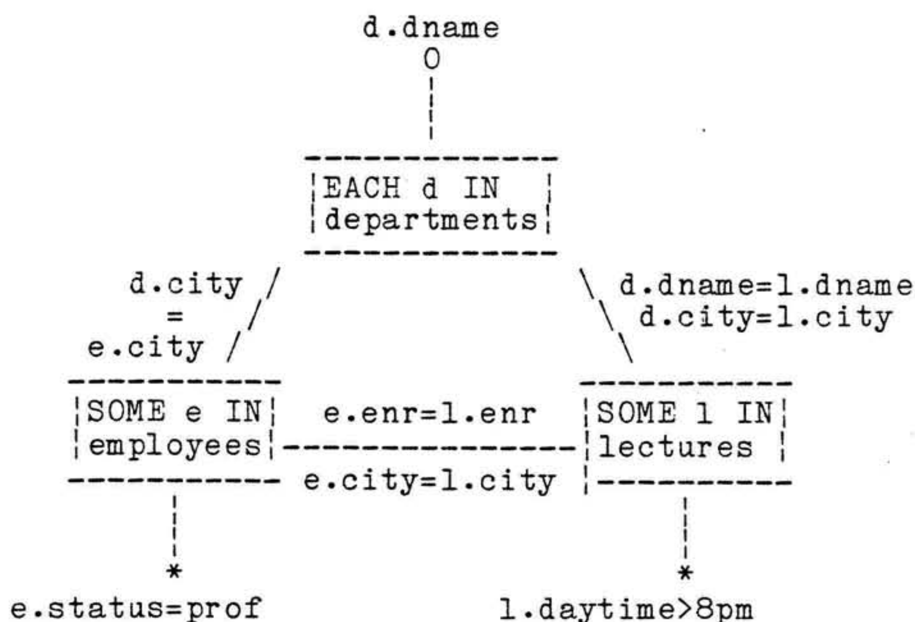

Figure 4.6: Augmented object graph for example 4.3

The additional terms 'd.city=l.city' and 'l.city=e.city' imply the condition 'd.city=e.city' by transitivity. Thus, the resulting graph is equivalent to a chain, a special form of tree. Note, that adding the new terms (conceptionally) requires to add the city attribute to the schema of the lectures relation (to be initialized with null values).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The query tree is then processed in semijoin fashion by executing a generalized semijoin for each edge, taking into account the newly introduced attributes. The amount of data transfer is reduced by means of specialized compression techniques.

Methods for the efficient implementation of operations like the ones presented in this section are candidates for hardware components to be integrated in specialized database machines. Since a detailed discussion of this topic would go beyond the scope of this paper, we shall only refer to a set of articles [LANG78], [SMIT79], [MARY80] containing a survey of hardware approaches to query optimization.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

5. ACCESS PLANS

The previous section dealt with techniques for the efficient evaluation of query components that can be used as building blocks of a general query evaluation algorithm. The question remains how to combine these blocks into an optimal or at least heuristically good evaluation procedure for arbitrary expressions. The input of such a procedure should be the logically pre-processed query as described in section 3, the existing storage structures and access paths, and a cost model. The output is an optimal access plan. The procedure consists of the following steps.

1. Generate all reasonable logical access plans for evaluating the query. Ameliorating transformations (section 3.3) may reduce the number of plans to be generated and compared.

2. Augment the logical access plans by details of the physical representation of data as gained from the meta database (sort orders, existence of physical access paths, statistical information).

3. Apply a model of access and processing costs to choose the cheapest access path.

An early example of such a process is described in [SELI79]. [YAO79] analyzes the optimal evaluation of two-variable quantifier-free queries in a framework similar to the one described above. A rather complete approach recently proposed by [ROSE82] will be described in some detail later. An example in a distributed environment is [BERN81a].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

This section first reviews the generation of access plans and then the choice problem and the related cost models. The quality of the optimal solution is strongly influenced by the available storage structures and access paths. They usually cannot be optimized for a single ad-hoc query. Therefore, the last subsection briefly considers the simultaneous optimization of multiple queries.

5.1 Generation of Access Plans

Access plans describe sequences of operations (operator graphs) or intermediate results (object graphs) leading from the existing data structures to a query result. A good query optimizer should generate a set of plans rich enough to contain the optimal plan but small enough to keep the optimization effort acceptable.

Two extreme approaches are described in [SMIT75] and [YAO79]. [SMIT75] use a rigid set of 'automatic programming' query transformation rules similar to the ones discussed in section 3. The procedure generates exactly one access plan which need not be optimal.

On the other hand, [YAO79] generates all access plans possible in a given physical environment. While this may be feasible in the context of two-variable queries, it becomes prohibitively costly for very complex queries.

Other approaches look for a compromise between heuristic selection and detailed generation of alternative access plans.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

For example, System R [ASTR75], [SELI79] applies a hierarchical procedure based on the nested block concept of SQL. On the lower level, evaluation plans for each query block are generated and compared. On the upper level, the sequence in which the query blocks are evaluated is determined. [KIM82] notes that this concept puts too much emphasis on the user-specified block structure of the query and therefore introduces query standardization steps.

A similar compromise was chosen in INGRES [WONG79], where the heuristic decomposition approach reduces a query to a set of subqueries containing at most two variables. For each of these subqueries, a more detailed analysis of its optimal implementation is performed.

A more comprehensive procedure for generating access plans to solve conjunctive queries without universal quantifiers and aggregates was proposed by [ROSE82]. In two steps, an object graph of logical access plans and an operator graph of physical access plans are developed.

The first step starts with a 'join template' of the query, that is, an object graph where the nodes describe range relations and the edges correspond to dyadic terms. A final node represents the query result. Each hierarchy with the final node as its root and existing data structures as leaf nodes represents a logical access plan. The restriction of range relations by monadic terms and the selection of target attributes by projection are not considered at this level, but are supposed to be taken care of in parallel to any access to the range relations.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

The join template closely corresponds to the pre-optimized form of a query as described in section 3. However, query evaluation can also make use of direct access structures present in the DBMS, for example, indexes or CODASYL set chains. The logical access plans are completed by augmenting the join template with sequences of intermediate results which make use of these existing access paths.

Some nodes in the logical representation are class nodes, that is, they represent a whole class of alternative physical representations or operation results. These are elaborated in the second part of the access plan generation procedure. [ROSE82] consider the operations of joining, scanning, sorting, and creating an access path. For joins and scans, alternative implementations are analyzed. The (usually very large) graph of physical operation sequences is not created in full since non-optimal paths to intermediate results are pruned as soon as better ones are detected.

5.2 Selection of Access Plans

The choice among physical access plans either follows heuristic rules or is based on a cost model of storage structures and access operations. In this subsection, cost models and their integration into optimization procedures will be reviewed.

While a few researchers consider working storage requirements [PALE72], [KIM82] or CPU costs [GOTL75], [SELI79] most cost models are based on the number of secondary storage accesses. For a given operation, this figure is influenced by the size of its operands, by the access structures used, and by the size of main memory buffers.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

At the beginning of the evaluation, the operands are existing data structures of known size such as relations or indexes. In later stages, however, most operands are results of preceding operations and the cost model must estimate their size using information about the original data structures and the selectivity of the operations already performed on them.

[DEMO80] and [RICH81] give comprehensive listings of size estimates for operations such as restriction, projection, and join based on certain assumptions. The review of these assumptions, below, is largely based on the work of [CHRI81].

The number of elements selected from a relation of size N by a condition A=c is N*f(A=c) where f(A=c) is called the restriction selectivity factor for attribute A and constant c. Most cost models assume a uniform distribution of attribute values, that is, f(A=c) = 1/n(A) where n(A) is the number of different values occuring in A. [CHRI81] has shown that this is not only unrealistic in many large databases, but also leads to very pessimistic cost estimates inhibiting the use of direct access structures. He therefore suggests more general assumptions about the value distributions, e.g., exponential distributions or a combination of a discrete distribution for the most frequent values and a simple assumption for the rest.

For example, in a relation describing university members, the number of elements of type 'student' (say 90% of all elements) can be stored explicitly while the uniform distribution assumption applies to the rest (professors, deans, secretaries, etc.).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Following an independence assumption of attribute value distributions, the selectivity factor for a conjunction of restrictions on different attributes is traditionally set to be the product of the selectivity factors for each restriction. Again, [CHRI81] has shown this to be pessimistic and presents estimates using knowledge about correlations between occurences of attribute values.

The number of elements in a equi-join can be estimated using a join selectivity factor j12 that estimates the number of pairs <x1,x2> satisfying the join condition to N1*N2*j12. To compute j12, it is usually assumed that the join field values have uniform distribution and that the two distributions are independent. Multivariate statistics can yield more realistic estimates [CHRI81].

The final cost measure is the number of secondary storage accesses not the sizes of intermediate results. The relationship between the two figures depends on the physical storage structures involved and on the proportion of elements to be accessed.

Assume first that all elements of an operand of size N have to be accessed. Then, the optimal number of secondary storage accesses would be N/B where B is the blocking factor of the operand. This can only be achieved if the elements are stored densely and if it is clear from the beginning on which physical records the elements reside.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

For example, the so-called segment scan of System R has to look at a superset of the necessary pages to find all elements of a relation [SELI79]. If the elements have to be read in some predetermined sequence, the elements must not only be stored densely but also sorted by the given reading order.

If direct access to a subset of the elements is used, the number of secondary storage accesses to retrieve n of the N elements depends on the clustering of elements on physical blocks. Optimal clustering can reduce the number of pages to be accessed to n/B, and the conventional random placement assumption [CHRI81] is a worst case one.

In conclusion, the traditional assumptions about value distributions and element placements tend to overestimate costs and thus to bias cost estimates against direct access structures. On the other hand, the more sophisticated techniques require more statistical information about the database. The question of how to keep such information up-to-date is not yet fully resolved.

How are the cost estimates used in query optimization? As mentioned in the previous subsection, there are heuristic procedures that do not use them at all. Other approaches combine heuristic reduction of choices with enumerative cost minimization in the 'end game' [WONG79]. Some experiments indicate that some combinatorial analysis can improve database performance considerably [EPST80].

For that part of the choice procedure that does make use of cost estimates, there are two ways to do so.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

First, the costs of each alternative access plan can be determined completely [BLAS76], [YAO79]. This approach has the advantage of covering techniques like parallelity or feedback in a realistic way. On the other hand, the optimization effort is high.

Second, the cost of strategies can be computed incrementally in parallel to their generation. While being sometimes less precise, this approach allows to evaluate whole families of strategies with common parts in parallel and thus reduces optimization costs considerably. For example, [ROSE82] keep only the cost minimal way to each intermediate result while discarding the rest as soon as its non-optimality is detected.

An extension of the second approach would be a dynamic query optimization procedure. The idea derives from the observation that, at each moment, only the next operation to be performed has to be finally determined. To guarantee overall optimality, only the consequences of this decision for the rest of the algorithm must be evaluated. A dynamic procedure has actual information about all its operands including intermediate results. This information can also be used to update the estimates of the remaining steps.

Besides the costs of such a procedure itself, there is a danger to get stuck in local optima if no lookahead is applied. However, if used carefully, it could improve the performance of the evaluation of such queries for which the actual intermediate results differ from the expected ones.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

5.3 Support for Multiple Queries

All query evaluation procedures presented in previous sections concentrate on optimizing the performance for a single query. This may be inefficient on the average when compared with a strategy that supports multiple queries simultaneously because such a strategy can consider "investments" in additional access paths that are not cost-effective for a single query. A limited number of approaches addresses this problem. They can be classified in four groups by the time scope for which decisions are made: (1) simultaneous optimization of batched queries; (2) context-sensitive query processing; (3) index selection; and (4) physical database design.

A set of queries which are submitted at approximately the same time can be batched for more efficient evaluation. The techniques for batched evaluation are similar to those described in subsection 4.3 for multi-variable expressions. For example, results of common subexpressions can be shared among queries, and subexpressions accessing the same physical data can do so in parallel. In addition, certain physical access paths such as sorting or temporary indexes can be provided which pay off for the batch as a whole. Little is known about detailed results in this area. [KIM81b] has developed a preliminary architecture, and a number of ongoing research projects is described in [DATA82].

Even if the queries to be optimized are not known in advance query processing strategies can at least make use of some known context in which queries are asked. One approach [CHAN79], [MAIE81], [FINK82] observes that queries often refer to former queries and

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

that it may be profitable to store query results, as derived relations or simply in a FIFO buffer. Another group of techniques makes use of language constructs such as selectors [MALL82] or views [ROUS82] to support queries in the context provided by these concepts.

Many examples in this paper show the importance of existing indexes for the performance of query evaluation algorithms. From this aspect, indexes can hardly hurt anywhere but are most profitable if they are very selective and support access to attributes frequently refered to in queries. However, index selection must also take into account altering transactions because they must change the index in addition to the base data. The index selection problem has been described in several survey [BATO82] and tutorial papers [SEVE77], [TEOR80]. The statistical assumptions discussed in the previous subsection also underly many of these models preventing the creation and use of possibly profitable indexes [CHRI81].

Finally, the efficiency of query optimization also depends on the general underlying database design. Important aspects include the horizontal clustering of relation elements by attribute values [SALT78], and the vertical partitioning of attributes by frequency of combined access [HAMM79]. However, longterm query optimization is only one of many aspects of physical database design. The general problem is so complex that decision support systems have been built to combine computerized optimization and human judgement [CARL81].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

SUMMARY AND CONCLUSIONS

An overview of logical transformation techniques and physical evaluation methods was given using the framework of the relational calculus. It was shown that a large body of knowledge has been developed to solve the problem of query evaluation efficiency in centralized databases.

Even disregarding the problems of distributed query optimization, however, many problems remain open. Promising research problems include: global optimization of complex queries with quantifiers and other aggregates; simultaneous optimization of multiple queries; and context-sensitive query processing including the use of database semantic. Another interesting area not directly addressed in this survey is query optimization in database systems with more advanced access paths such as multiple attribute indexes and database machines, or with complex data structures such as statistical databases, historical databases, or CAD/CAM databases.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

REFERENCES

Query Optimization Bibliography

AHO79a Aho, A.V., Sagiv, Y., Ullman, J.D. "Efficient Optimization of a Class of Relational Expressions", ACM Trans. Database Syst. 4, 4 (Dec. 1979), 435-454.

AHO79b Aho, A.V., Beeri, C., Ullman, J.D. "The Theory of Joins in Relational Databases", ACM Trans. Database Syst. 4, 3 (Sept. 1979), 297-314.

AHO79c Aho, A.V., Sagiv, Y., Ullman, J.D. "Equivalences among Relational Expressions", SIAM J. Comptg. 8, 2 (May 1979), 218-246.

ASHA78 Ashany, R. "Application of Sparse Matrix Techniques to Search, Retrieval, Classification and Relationship Analysis in Large Data Base Systems - SPARCOM", Proc. 4th VLDB Conf., West Berlin, 1978, 499-516.

ASTR74 Astrahan, M.M., Ghosh, S.P. "A Search Path Selection Algorithm for the Data Independent Accessing Model (DIAM)", Proc. ACM-SIGMOD Conf., Ann Arbor, Michigan, May 1974, 367-388.

ASTR75 Astrahan, M.M., Chamberlin, D.D. "Implementation of a Structured English Query Language", Comm. ACM 18, 10 (Oct. 1975), 580-588.

BATO82 Batory, D.S. "Index Selection", in Yao, S.B. (ed.) Principles of Database Design, Springer 1982.

BERN81a Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L., Rothnie, J.R. "Query Processing in a System for Distributed Databases (SDD-1)", ACM Trans. Database Syst. 6, 4 (Dec. 1981), 602-625.

BERN81b Bernstein, P.A., Goodman, N. "The Power of Natural Semijoins", SIAM J. of Comptg. 10, 4 (Nov. 1981).

BERN81c Bernstein, P.A., Goodman, N. "The Power of Inequality Semijoins", Inform. Systems 6, 4 (May 1981), 255-265.

BERN81d Bernstein, P.A., Chiu, D.M. "Using Semi-Joins to Solve Relational Queries", J.ACM 28, 1 (Jan. 1981), 25-40.

BERN79a Bernstein, P.A., Goodman, N. "Inequality Semi-Joins", Technical Report CCA-79-28, Cambridge, 1979.

BERN79b Bernstein, P.A., Goodman, N. "The Theory of Semi-Joins", Technical Report CCA-79-27, Cambridge, 1979.

BLAS77 Blasgen, M.W., Eswaran, K.P. "Storage and Access in Relational Databases", IBM Syst. J. 16 (1977), 363-377

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

BLAS76 Blasgen, M.W., Eswaran, K.P. "On the Evaluation of Queries in a Relational Data Base System", IBM Research Report RJ 1745, April 1976.

BREI75 Breitbart, Y., Reiter, A. "Algorithms for Fast Evaluation of Boolean Expressions", Acta Informatica 4, 1975, 107-116.

BUNE79 Buneman, P. "The Problem of Multiple Paths in a Database Schema", Proc. 5th VLDB Conf., Rio de Janeiro, 1979, 368-372.

CARL76 Carlson, C.R., Kaplan, R.S. "A Generalized Access Path Model and its Application to Relational Data Base System", Proc. ACM-SIGMOD Conf., Washington D.C., 1976, 143-154.

CHAM79 Chamberlin, D.D., Astrahan, M.M., Lorie, R.A., Mehl, J.W., Price, T.G., Schkolnick, M., Selinger, P.G., Slutz, D.R., Wade, B.W., Yost, R.A. "Support for Repetitive Transactions and ad-hoc Queries in System R", IBM Research Report RJ2551, May 1979.

CHAN77 Chandra, A.K., Merlin, P.M. "Optimal Implementation of Conjunctive Queries in Relational Data Bases", Proc. 9th Annual ACM Symp. on Theory of Computation, Boulder, Col., 1977.

CHAN79 Chang, C.L. "On Evaluation of Queries Containing Derived Relations in a Relational Data Base", IBM Research Report RJ2667, October 1979.

CHIU81 Chiu, D.M., Bernstein, P.A., Ho, Y.C. "Optimizing Chain Queries in a Distributed Database System", Technical Report TR-01-81, Harvard University, 1981.

CHIU80 Chiu, D.M., Ho, Y.C. "A Methodology for Interpreting Tree Queries Into Optimal Semi-Join Expressions", Proc. ACM-SIGMOD Conf., Santa Monica, 1980, 169-178.

CHRI81 Christodoulakis, S. "Estimating Selectivities in Data Bases", Technical Report CSRG-136, Univ. of Toronto, 1981.

CLAU80 Clausen, S.E. "Optimizing the Evaluation of Calculus Expressions in a Relational Database System", Inform. Systems 5, 1980, 41-54.

CODD72 Codd, E,F. "Relational Completeness of Data Base Sublanguages", in Courant Computer Science Symposia, No. 6, Data Base Systems (New York City, May 1971), Prentice Hall.

DATA82 Special Issue on Query Optimization, Database Engineering 5, 3 (September, 1982).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

DAYA82 Dayal, U., Goodman, N. "Query Optimization for CODASYL Database Systems", Proc. ACM-SIGMOD Conf., Orlando, 1982, 138-150.

DEMO80 Demolombe,, R. "Estimation of the Number of Tuples Satisfying a Query Expressed in Predicate Calculus Language", Proc. 6th VLDB Conf., Montreal, 1980, 55-63.

DEWI79 DeWitt, D.J. "Query Execution in DIRECT", Proc. ACM-SIGMOD Conf., Boston, 1979, 13-22.

EPST78 Epstein, R., Stonebraker, M., Wong, E. "Distributed Query Processing in a Relational Data Base System", ACM-SIGMOD Conf., Austin, Texas, May 1978.

EPST80 Epstein, R. "Analysis of Distributed Data Base Processing Strategies", Memorandum No. UCB/ERL M80/25, Univ. of California, Berkeley, 1980.

FINK82 Finkelstein, S. "Common Expression Analysis in Database Applications", Proc. ACM-SIGMOD Conf., Orlando, 1982, 235-245.

GILL75 Gilles, J.H., Schuster, S.A. "Query Execution and Index Selection for Relational Data Bases", Techn. Report, CSRG-53, University of Toronto, 1975.

GOOD82 Goodman, N., Shmueli, O. "The Tree Property is Fundamental for Query Processing", Proc. ACM Symp. on Principles of Database Systems", Los Angeles, 1982, 40-48.

GOOD81 Goodman, N., Shmueli, O. "Nonreducible Database States for Cyclic Queries", Techn. Report, TR-15-80, Harvard Univ., July 1980.

GOTL75 Gotlieb, L.R. "Computing Joins of Relations", Proc. ACM-SIGMOD Conf., San Jose, 1975.

GOUD81 Gouda, M.G., Dayal, U.D. "Optimal Semijoin Schedules for Query Processing in Local Distributed Database Systems", in Proc. ACM-SIGMOD Conf., April 1981, 164-175.

GRIF78 Griffeth, N.D. "Nonprocedural Query Processing for Databases With Access Path", ACM-SIGMOD Conf., Austin, Texas, May 1978.

GRIS78 Grishman, R. "The Simplification of Retrieval Requests Generated by Question-Answering Systems", Proc. 4th VLDB Conf., West Berlin, 1978, 400-406.

GUDE73 Gudes, E., Reiter, A. "On Evaluating Boolean Expressions", Software-Practice and Experience 3 (1973), 345-350.

HALL76 Hall, P.A.V. "Optimization of Single Expressions in a Relational Data Base System", IBM J.Res. Develop. 20(3), 1976, 244-257.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

HALL74a Hall, P.A.V. "Common Subexpression Identification in General Algebraic Systems", Techn. Rep. UKSC 0060, IBM UKSC, Peterlee, 1974.

HALL74b Hall, P.A.V., Todd, S.J.P. "Factorization of Algebraic Expressions", Techn. Rep. UKSC 0055, IBM UKSC, Peterlee, 1974.

HAMM80 Hammer, M., Zdonik, S. "Knowledge-Based Query Processing", Proc. 6th VLDB Conf., Montreal 1980, 137-147

HANA77 Hanani, M.Z. "An Optimal Evaluation of Boolean Expressions in an On-Line Query System", CACM 20, 5 (1977), 344-347.

HARD79 Hardgrave, W.T. "Ambiguity in Processing Boolean Queries on TDMS Tree Structures: A Study of Four Different Philosophies", Proc. 5th VLDB Conf., Rio de Janeiro, 1979, 373-397.

HEVN79 Hevner, A.R., Yao, S.B. "Query Processing on a Distributed Database", IEEE Trans. Softw. Eng. SE-5, 3 (May 1979), 177-187.

JARK82a Jarke, M., Schmidt, J.W. "Query Processing Strategies in the PASCAL/R Relational Database Management System", Proc. ACM-SIGMOD Conf., Orlando, 1982, 256-264.

JARK81 Jarke, M., Schmidt, J.W. "Evaluation of First-Order Relational Expressions", Technical Report No. 78, Universitaet Hamburg, Fachbereich Informatik, June 1981.

KAMB82 Kambayashi, Y., Yoshikawa, M., Yajima, S. "Query Processing for Distributed Databases Using Generalized Semi-Joins", Proc. ACM-SIGMOD Conf., Orlando, 1982, 151-160.

KERS80a Kershberg, L., Ting, P.D., Yao, S.B. "Optimal Distributed Query Processing", Technical Report, Bell Labs Holmdel, 1980.

KERS80b Kershberg, L., Ting, P.D., Yao, S.B. "Query Optimization in Star Computer Networks", Techn. Report, Bell Labs Holmdel, March 1980.

KING81 King, J.J. "QUIST: A System for Semantic Query Optimization in Relational Data Bases", Proc. 7th VLDB Conf., Cannes, 1981, 510-517.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

KIM82 Kim, W. "On Optimizing an SQL-like Nested Query", ACM Trans. Database Syst. 7, 3 (September 1982), 443-469.

KIM81a Kim, W. "A Bit-Serial/Tuple-Parallel Relational Query Processor", IBM Research Report RJ3194, July 1981.

KIM81b Kim, W. "Query Optimization for Relational Database Systems", IBM Research Report RJ3081, March 1981.

KIM80 Kim, W. "A New Way to Compute the Product and Join of Relations", Proc. ACM-SIGMOD Conf., Santa Monica, 1980.

KLUG82a Klug, A. "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", J. ACM 29, 3 (July 1982), 699-717.

KLUG82b Klug, A. "Access Paths in the 'ABE' Statistical Query Facility", Proc. ACM-SIGMOD Conf., Orlando, 1982, 161-173.

LEIL78 Leilich, H.-O., Stiege, G., Zeidler, H. Ch. "A Search Processor for Data Base Management Systems", Proc. 4th VLDB Conf., West Berlin, 1978, 280-287.

LIU76 Liu, J.W. "Algorithms for Parsing Search Queries in Systems with Inverted File Organizations", ACM Trans. Database Syst. 1, 4 (Dec. 1976).

LOZI80 Lozinski, E.L. "Construction of Relations in Relational Databases", ACM Trans. Database Syst. 5, 2 (June 1980), 208-224.

MAKI81 Makinouchi, A., Tezuka, M., Kitakami, H., Adachi, S. "The Optimization Strategy for Query Evaluation in RDB/V1", Proc. 7th VLDB Conf., Cannes, 1981, 518-531.

MAIE81 Maier, D., Warren, D.S. "Incorporating Computed Relations in Relational Databases", Proc. ACM-SIGMOD Conf., Ann Arbor, 1981.

MERR81a Merrett, T.H. "Why Sort-Merge Gives the Best Implementation of the Natural Join", Techn. Report SOCS-81-37, McGill University, Montreal, 1981.

MERR81b Merrett, T.H., Kambayashi, Y., Yasuura, H. "Scheduling of Page-Fetches in Join Operations", Proc. 7th VLDB Conf., Cannes, 1981, 488-498.

MERR81c Merrett, T.H. "Practical Hardware to Linear Execution of Relational Database Operations", Techn. Report SOCS-81-30, McGill University, Montreal, Sept. 1981.

NIEB76a Niebuhr, K.E., Smith, S.E. "N-ary Join for Processing Query by Example", IBM Technical Disclosure Bulletin, Vol. 19, No. 6, November 1976, 2377-2381.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

NIEB76b Niebuhr, K.E., Scholz, K.W., Smith, S.E. "Algorithm for Processing Query by Example", IBM Technical Disclosure Bulletin, Vol. 19, No. 2, July 1976, 736-741.

PALE72 Palermo, F.P. "A Data Base Search Problem", Proc 4th Comp. and Inform. Sc. Symp., Miami Beach, 1972, 67-101.

PAPA82 Papakonstantinou, P. "Optimal Evaluation of Queries", The Computer Journal 25, (1982), 237-241.

PECH76 Pecherer, R.M. "Efficient Exploration of Product Spaces", Proc. ACM-SIGMOD Conf., Washington D.C., 1976, 169-177.

PUTK79 Putkonen, A. "On the Selection of the Access Path in Inverted Database Organizations", Inform. Systems 4, 1979, 219-225.

RICH81 Richard, P. "Evaluation of the Size of a Query Expressed in Relational Algebra", Proc. ACM-SIGMOD Conf., Ann Arbor, Michigan, May 1981, 155-163.

ROSE80 Rosenkrantz, D.J., Hunt, M.B. "Processing Conjunctive Predicates and Queries", Proc. 6th VLDB, Montreal, 1980, 64-74.

ROSE82 Rosenthal, A., Reiner, D. "An Architecture for Query Optimization", Proc. ACM-SIGMOD Conf., Orlando, 1982, 246-255.

ROTH75 Rothnie, J.B. "Evaluating Inter-Entry Retrieval Expressions in a Relational Data Base Management System", Proc. National Computer Conf., 1975, 417-423.

ROTH74 Rothnie, J.B. "An Approach to Implementing a Relational Data Management System", Proc. ACM-SIGMOD Conf., 1974, 277-294.

ROUS82 Roussopoulos, N. "View Indexing in Relational Databases", ACM Trans. Database Syst. 7, 2 (June 1982).

SAGA77 Sagalowicz, D. "IDA: An Intelligent Data Access Program", Proc. 3th VLDB Conf., Tokyo, 1977, 293-302.

SAGI80 Sagiv, Y., Yannakakis, M. "Equivalences among Relational Expressions with the Union and Difference Operators", JACM 27 (1980), 633-655.

SAGI81 Sagiv, Y. "Optimization of Queries in Relational Databases", Computer Science: Distributed Database Systems, No 12, UMI Research Press, Ann Arbor, Michigan, 1981.

SALT78 Salton, G. "Generation and Search of Clustered Files", ACM Trans. Database Syst. 3, (1978), 321-346.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

SCHE77 Schenk, K.L., Pinkert, J.R. "An Algorithm for Servicing Multi-Relational Queries", Proc. ACM-SIGMOD Conf., Toronto, 1977, 10-19.

SCHM79 Schmidt, J.W. "Parallel Processing of Relations: A Single-Assignment Approach", Proc. 5th VLDB, Rio de Janeiro, 1979, 398-408.

SELI79 Selinger, P., Astrahan, M.M., Chamberlin, D.D., Lorie, P.A., Price, T.G. "Access Path Selection in a Relational Database Management System", Proc. ACM-SIGMOD Conf., Boston, 1979.

SEVE77 Severance, D.G., Carlis, J.V. "A Practical Approach to Selecting Record Access Paths", Computing Surveys 9, (1977), 259-272.

SHMU81 Shmueli, O. "The Fundamental Role of Tree Schemas in Relational Query Processing", PhD thesis, Harvard Univ., Aug. 1981.

SMIT75 Smith, J.M., Chang, P.Y.T. "Optimizing the Performance of a Relational Algebra Database Interface", Comm. ACM 18, 10 (Oct. 1975), 568-579.

STRO79 Stroet, J.W.M., Engmann, R. "Manipulation of Expressions in a Relational Algebra", Inform. Systems 4, 1979, 195-203.

TODD74 Todd, S. "Implementing the Join Operator in Relational Data Bases", IBM Scientific Center Technical Note 15, IBM UK Scientific Center, Peterlee, England, Nov. 1974.

WALK80 Walker, A. "On Retrieval from a Small Version of a Large Database", Proc. 6th VLDB Conf., Montreal, 1980, 47-54.

WARR81 Warren, D.H.D. "Efficient Processing of Interactive Relational Data Base Queries Expressed in Logic", Proc. 7th VLDB Conf., Cannes, 1981, 272-283.

WELC76 Welch, J.W., Graham, J.W. "Retrieval Using Ordered Lists in Inverted and Multilist Files", Proc. ACM-SIGMOD Conf., Washington D.C., 1976, 21-30.

WONG76 Wong, E., Youssefi, K. "Decomposition - A Strategy for Query Processing", ACM Trans. Database Syst. 1, 3 (Sept. 1976), 223-241.

WONG79 Wong, E., Youssefi, K. "Query Processing in a Relational Database Management System", Proc. 5th VLDB Conf., Rio de Janeiro, 1979, 409-417.

YAO79 Yao, S.B. "Optimization of Query Evaluation Algorithms", ACM Trans. Database Syst. 4, 2 (June 1979), 133-155.

YAOD78 Yao, S.B., DeJong, D. "Evaluation of Database Access Paths", Proc. ACM-SIGMOD Conf., Austin, 1

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

Other References Appearing in the Article

BAYE72 Bayer, R., McCreight, E. "Organization and Maintenance of Large Ordered Indexes", Acta Informatica 1 (1972), 173-189.

BENT79 Bentley, J.L., Friedman, J.H. "Data Structures for Range Searching", Computing Surveys 11 (1979), 397-412.

BERN81e Bernstein, P.A., Goodman, N. "Concurrency Control in Distributed Database Systems", Computing Surveys 13, 2 (1981).

CARL81 Carlis, J.V., March, S.T., Dickson, G.W. "Physical Database Design: A DSS Approach", Proc. 2nd Information Systems Conf., Boston, 1981, 153-172.

CHAM76 Chamberlin, D.D. "Relational Data-Base Management Systems", Computing Surveys 8, 1 (March 1976).

GRIE71 Gries, D. "Compiler Construction for Digital Computers", New York - London, J. Wiley & Sons, 1971.

HAMM79 Hammer, M., Niamir, B. "A Heuristic Approach to Attribute Partitioning", Proc. ACM-SIGMOD Conf., Boston, 1979, 93-101.

IBM66 IBM Corp. "Introduction to IBM Direct-Access Storage Devices and Organization Methods", GC 20-1649-06, 1966.

JARK82b Jarke, M., Vassiliou, Y. "Choosing a Database Query Language", submitted for publication, 1982.

KIM79 Kim, W. "Relational Database Systems", Computing Surveys 11, (1979), 185-212.

LANG78 Langdon, J.J. "A Note on Associative Processors for Data Management", ACM Trans. Database Systems 3 (1978), 148-158.

MALL82 Mall, M., Reimer, M., Schmidt, J.W. "Data Selection, Sharing and Access Control in a Relational Scenario", in "Perspectives on Conceptual Modelling", Springer Verlag, 1982.

MARY80 Maryanski, F.J. "Backend Database Systems", Computing Surveys 12, (1980), 3-26.

MERR77 Merrett, T.M. "Database Cost Analysis: A Top-Down Approach", Proc. ACM-SIGMOD Conf., Toronto, 1977, 135-143.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73

PIRO79 Pirotte, A. "Fundamental and Secondary Issues in the Design of Non-Procedural Relational Languages", Proc. VLDB Conf., Rio de Janeiro, October 1979, 239-250.

SOCK79 Sockut, G.H., Goldberg, R.P. "Database Reorganization - Principles and Practice", Computing Surveys 11, (1979), 371-396.

SCHM77 Schmidt, J.W. "Some High Level Language Constructs for Data of Type Relation", ACM Trans. Database Syst. 2, 3 (Sept. 1977).

SMIT79 Smith, D., Smith, J. "Relational Database Machines", IEEE Computer, 12, 3 (March 1979), 28-38.

TAYL76 Taylor, R.W., Frank, R.L. "CODASYL Data-Base Management Systems", Computing Surveys 8, 1 (March 1976).

TEOR80 Teorey, T.J., Fry, J.P. "The Logical Record Access Approach to Database Design", Computing Surveys 12, (1980), 179-212.

TSIC76 Tsichritzis, D.C., Lochovsky, F.H. "Hierarchical Data-Base Management Systems", Computing Surveys 8, 1 (March 1976).

VASS82 Vassiliou, Y., Jarke, M. "Query Languages - A Taxonomy", NYU Symposium on User Interfaces, New York, 1982, Ablex Publ. Corp. (to appear).

VERH78 Verhofstad, J.S.M. "Recovery Techniques for Database Systems", Computing Surveys 10, (1978), 167-197.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-82-73