Query Optimization in Database Systems

MATTHIAS JARKE

Graduate School of Business Administration, New York University, New York, New York 10006

JÜRGEN KOCH

Fachbereich Informatik, Johann Wolfgang Goethe-Universität, 6000 Frankfurt 1, West Germany

Efficient methods of processing unanticipated queries are a crucial prerequisite for the success of generalized database management systems. A wide variety of approaches to improve 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 alternative access plans and choosing among them.

These methods are presented in the framework of a general query evaluation procedure using the relational calculus representation of queries. In addition, nonstandard query optimization issues such as higher level query evaluation, query optimization in distributed databases, and use of database machines are addressed. The focus, however, is on query optimization in centralized database systems.

Categories and Subject Descriptors: A.1 **[General Literature]**: Introductory and Survey; H.2.2 **[Database Management]**: Physical Design—*access methods*; H.2.3 **[Database Management]**: Languages—*query languages*; H.2.4 **[Database Management]**: Systems—*query processing*; H.2.6 **[Database Management]**: Database Machines; I.1.1 **[Algebraic Manipulation]**: Expressions and Their Representation—*simplification of expressions*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Database implementation, query optimization, query simplification

INTRODUCTION

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

There have been two major areas of research in database systems. One is the analysis of data models into which the real world can be mapped and on which interfaces for different user types can be built. Such conceptual models include the hierarchical, the network, the relational, and a number of semantics-oriented models that have been reviewed in a large number of books and surveys [Brodie et al. 1984].

A second area of interest is the safe and efficient implementation of the DBMS. Computerized data have become a central resource of most organizations. Each implementation meant for production use must take this into account by guaranteeing the safety of the data in the cases of concurrent access [Bernstein and Goodman 1981c], recovery [Verhofstad 1978],

CONTENTS

and reorganization [Sockut and Goldberg 1979]. One major criticism of many early DBMSs has been their lack of efficiency in handling the powerful operations they offer, particularly the content-based access to data by queries. *Query optimization* tries to solve this problem by integrating 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.

Traditionally, each of these approaches has used a different language. This is probably one of the reasons why no comprehensive survey of query optimization techniques has yet been presented. The goal of this paper is to review query optimization techniques in the common framework of relational calculus. This has been shown to be technically equivalent to a relational algebra representation [Codd 1972; Klug 1982a] and extendable to the implementation of network DBMSs [Dayal and Goodman 1982]. Moreover, many popular query languages, such as SQL [Astrahan and Chamberlin 1975] or QUEL [Stonebraker et al. 1976], map easily into relational calculus.

In the interest of space, the focus of the paper is primarily on the problem of optimizing queries in the centralized DBMS. Centralized query optimization is not only important in many mainframe databases—and more recently in microcomputer DBMSs—but also appears as a subproblem of query optimization in distributed systems. Distributed query optimization itself [Bayer et al. 1984; Sacco and Yao 1982; Ullman 1982] is only addressed briefly, and the following two related areas are not treated at all:

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 interface in the two areas consists of the functional capabilities and usability of the query language [Vassiliou and Jarke 1984], mainly in the response time of the system. If one assumes given functional capabilities of the query language and a response time minimization goal of the query evaluation 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 and the derivation of the related cost functions (see, e.g., Teorey and Fry [1982]) are beyond the scope of this paper.

The paper is organized into six sections, following a top-down approach. In Section 1 we present a global framework for query optimization. In Section 2 we compare four techniques for representing queries in terms of their suitability for optimization. In Section 3 we utilize one of these techniques, the relational calculus, for presenting logic-based transformations, including the emerging methods of semantic query optimization.

After being transformed, a query must be mapped into a sequence of operations that return the requested data. In Section 4 we analyze the implementation of such operations on a low-level system of stored data and access paths. In Section 5 we present optimization procedures for integrating these operations into a globally optimal access plan.

A number of query optimization problems require special treatment because of higher query complexity or certain characteristics of the underlying hardware. Three such problem areas—higher level queries, distributed queries, and queries using database machines—are summarized in Section 6.

1. THE QUERY OPTIMIZATION PROBLEM

Exact optimization of query evaluation procedures is in general computationally intractable and is hampered further by the lack of precise statistical information about the database. Query evaluation algorithms must rely heavily on heuristics. Nevertheless, the term "query optimization" will be used to refer to strategies intended to improve the efficiency of query evaluation procedures. In this section we state the objectives of query optimization and present a general procedure designed 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 mostly set-oriented) manner, giving the optimizer sufficient choices among alternative evaluation procedures.

Queries are used in several settings. The most obvious application is that of direct requests by end users who need information about the structure or content of the database. If the requests are limited to a set of standard queries, they can be optimized manually by programming the associated search procedures and restricting the user's input to a menu format. However, an automatic query optimization system becomes necessary if ad hoc queries are to be asked by use of a general-purpose query language.

A second application of queries occurs in transactions that change the stored data based on their current value (e.g., "give all assistant professors a 10 percent salary increase"). Finally, querylike expressions can be used internally in a DBMS, for example, to check access rights [Griffiths and Wade 1976], maintain integrity constraints [Stonebraker 1975], and synchronize concurrent accesses correctly [Reimer 1983].

1.2 Optimization Objectives

The economic principle requires that optimization procedures either attempt to maximize the output for a given number 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 in a given system environment. This general goal allows a number of different operational objective functions. The response time goal is reasonable only under the assumption that user time is the most important bottleneck resource. Otherwise, direct cost minimization of technical resource usage can be attempted. Fortunately, both objectives are largely complementary; when goal conflicts arise, they are typically resolved by assigning limits to the availability of technical resources (e.g., those of main memory buffer space).

In order to allow a fair comparison of efficiency, the functional capabilities of the query evaluation systems to be compared must be similar. The requirement of "relational completeness" coined by Codd [1972] (compare Section 2.1) has become a quasi-standard. The techniques surveyed in this paper are presented as contributions to the implementation of queries in a relationally complete language with minimal evaluation cost or response time. Queries of higher complexity [Chandra and Harel 1982a] are considered in Section 6.1. The total cost to be minimized is the sum of the following:

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 costs for the delay in processing caused by transmission. The latter, which is more important for query optimization, is often assumed to be a linear function of the number of data transmitted.

Secondary Storage Access Cost: The cost of (or time for) loading data pages from secondary storage into main memory. This is influenced by the number of data to be retrieved (mainly by the size of intermediate results), the clustering of data on physical pages, the size of the available buffer space, and the speed of the devices used.

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 if it can be varied from query to query.

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

The structure of query optimization algorithms is strongly influenced by the trade-off among these cost components. In long-range distributed DBMSs with relatively slow communication lines, communication delay dominates the costs, whereas the other factors are relevant only for local suboptimization. In centralized systems, the costs are dominated by the time for secondary storage accesses although the CPU costs may be quite high for complex queries [Gotlieb 1975]. In locally distributed DBMSs, all factors have similar weights, which results in very complex cost functions and optimization procedures.

Since the focus of this paper is on centralized databases, comuunication costs are not considered because in such systems communication requirements are independent of the evaluation strategy. For the optimization of single queries, storage costs are usually also assumed to be of secondary importance. They are considered only for the simultaneous optimization of multiple queries.

There remain the costs of secondary storage accesses (usually measured by the number of page accesses) and CPU usage (often measured by the number of comparisons to be performed). A number of common ideas underly most techniques developed to reduce these costs. They try to (1) avoid duplication of effort, (2) use standardized parts, (3) look ahead in order to avoid unnecessary operations, (4) choose the cheapest way to execute elementary operations, and (5) sequence them in an optimal fashion. The following simple example demonstrates what can be expected from query optimization.

Consider the relational schema of a database that describes employees offering computer lectures to departments of a geographically distributed organization:

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

Key attributes are underlined; a given combination of key attribute values identifies a relation element uniquely. Assume that a user is interested in the

> "names of departments located in New York offering courses on database management."

There are many possible strategies to solve this query, three of which are compared with respect to the following assumptions on actual data values. Note that the detailed data used for the computations below are not usually available to the query optimizer, but have to be estimated.

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

There are 500 "courses", 20 of which are on database management. The physical block size is 10 records.

There are 2000 "lectures". Three hundred are on database management, 100 are held in New York departments, and 20 (from 3 departments) satisfy both conditions. The physical block size is 10 records.

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. Finally, all relations are physically sorted by ascending key values.

The first strategy presented here follows a straightforward approach of translating a relational calculus expression into a sequence of algebra operations [Codd 1972]. Together with each step of the strategy, the numbers of secondary storage accesses required to read (r) or write (w) a physical block are given:

Strategy 1

1. Form the Cartesian product of the "courses", "lectures", and "departments" relations, and (r: 200000)
2. Retain the dname column of those "department" records, for which

   the cnr of "courses" and "lectures" match, and
   the dname of "lectures" and "departments" match,
   and cname = 'database management'
   and city = New York. (w: 1)

total: approximately 200000 accesses.

The extremely high cost of this strategy results from the fact that it generates an intermediate result, of which only a very small portion is actually relevant for further processing. Efficiency can be improved substantially by considering only those combinations of elements from different relations that have matching values in common attributes. By making use of existing sort orders, such combinations can be formed by "merging" the participating relations. This operation is called a "join":

Strategy 2

1. Merge "courses" and "lectures". (r: 50 + 200; w: 400)
2. Sort the result by dnames. (r + w: 400 log(2)400)
3. Merge the result with "departments". (r: 400 + 20; w: 400 + 400)
4. Select the combinations with city = 'New York' and cname = 'database management', and (r: 800)
5. keep only the dname column. (w: 1)

total: approximately 6000 accesses.

The cost for answering the query can be reduced further by performing value-based selections as early as possible and thereby reducing the cost for sorting and merging intermediate results:

Strategy 3

1. Merge "courses" with "lectures", and (r: 50 + 200)
2. keep only the dnames of combinations with cname = 'database management' (w: 2)
3. Sort the dname list generated. (r + w: 2)
4. Merge the result with the "departments" relation, and (r: 2 + 20)
5. keep only those dnames with city = 'New York'. (w: 1)

total: 277 accesses.

Thus a reduction by a factor of approximately 700 has been achieved. For larger databases and more complex queries, more sophisticated techniques may result in even higher reductions.

1.3 Top-Down Approach to Query Optimization

Query optimization research in the literature can be divided in two classes, which can be described as bottom up and top down. Researchers found the overall query optimization problem to be very complex. Theoretical work began with a bottom-up approach, studying special cases, such as the optimal implementation of important operations and evaluation strategies for certain simple subclasses of queries. Subsequently researchers attempted to compose larger building blocks from these early results.

A need for working systems triggered the development of full-scale query evaluation procedures, which stressed the generality of solutions and handled query optimization in a uniform and heuristic manner [Astrahan and Chamberlin 1975; Makinouchi et al. 1981; Niebuhr et al. 1976; Palermo 1972; Schenk and Pinkert 1977; Wong and Youssefi 1976]. As this often did not achieve competitive system efficiency, the current trend seems to be a top-down approach that incorporates more knowledge

about special case optimization opportunities into the general procedures. At the same time, the general algorithms themselves have been augmented by combinatorial cost-minimization procedures for choosing among strategies.

This paper follows the top-down approach, utilizing the general evaluation procedure that follows 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 that leaves the system all necessary degrees of freedom to optimize the evaluation.

Step 2. Apply logical transformations to the query representation that (1) standardize the query, (2) simplify the query to avoid duplication of effort, 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 sequences of elementary operations for which a good implementation and its associated cost 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 often can be handled at compile time. The quality of Steps 3 and 4, that is, the richness of the access plans generated and the optimality of the choice algorithm, heavily depends upon knowledge about the values in the database.

The consequences of data dependence are twofold. First, if the database is volatile, Steps 3 and 4 can be done only at run time. This means that the possible gain in efficiency must be traded off against the cost of the optimization itself. Second, a metadatabase (e.g., an augmented data dictionary) must maintain general information about the database structure as well as statistical information about the database contents. As in many similar operational research problems (e.g., inventory control), the costs of obtaining and maintaining this additional information must be compared to its value.

2. QUERY REPRESENTATION

Queries can be represented in a number of forms. In the context of query optimization, an appropriate query representation form must fulfill the following requirements: It should be powerful enough to express a large class of queries, and it should provide a well-defined basis for query transformation. In this section we 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 (tuple) relational calculus as introduced by Codd [1971, 1972] 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: 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 [Schmidt 1977].

Example 2.1. Names of processors who published some paper in 1981.

```
[⟨e.ename⟩ 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, that is, 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 target attribute specification

"⟨e.ename⟩" indicates that only the names of employees are retained for the query result.

The selection expression—the predicate following the colon—defines constraints on the free variable. The first constraint is a restrictive or *monadic term*, restricting the free variable to those "employees" records that have the status value "professor". This constraint is AND-connected with a join or *dyadic term*, relating "employees" to "papers", and another monadic term, further restricting the result to those employees who published some paper in 1981. The comparison operators usually allowed in terms are $=, \neq, <, >, \leq$, and $\geq$.

In contrast to the one-sorted predicate calculus, the relational calculus allows variables to be bound to different sorts (range relations); for instance, 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 3.1.

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. Atomic predicates:
   (i) A (monadic or dyadic) term is an atomic predicate.
   (ii) TRUE is an atomic predicate.
   (iii) FALSE is an atomic predicate.
2. An atomic predicate is a predicate. Let A be a predicate, r an element variable, and rel a relation. Then
   (i) SOME r IN rel(A),
   (ii) ALL r IN rel(A)

   are also 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 formulas are predicates.

In Codd [1972] the relation 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 that goes beyond relational completeness is "find the names of employees reporting to manager Smith at any level", provided that a hierarchy of employees is modeled in a single relation (e.g., via a name and manager attribute) [Pirotte 1979]. Furthermore, queries in today's applications often contain aggregations that cannot be expressed in pure relational calculus. However, the extension of relational calculus by aggregate functions is rather straightforward [Klug 1982b; Maier and Warren 1981].

2.2 The Relational Algebra

The relational algebra as defined by Codd [1972] is a collection of operators on relations. These operators fall into two classes, that is, 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 are defined below by relating them to equivalent relational calculus expressions.

The *restriction* operator applied to a relation "rel" constructs a horizontal subset according to a quantifier-free predicate containing only monadic terms or intrarelational dyadic terms (comparisons between two attributes of the same relation element):

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

The *projection* operator serves to construct a vertical subset of a relation "rel" by selecting a set A of specified attributes and eliminating duplicate tuples within these attributes:

Proj(rel, A) = [⟨$r.A$⟩ OF EACH r IN rel: true].

The *join* operator permits two relations "rel_1" and "rel_2" to be combined into a

single relation whose attributes are the union of the attributes of "rel_1" and "rel_2":

Join(rel_1, A op B, rel_2)
= [EACH r_1 IN rel_1, EACH r_2 IN rel_2:
$r_1.A$ op $r_2.B$].

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(rel_1, A by B, rel_2)
= [⟨r.compl(A)⟩ OF EACH r_1 IN rel_1:
ALL r_2 IN rel_2 SOME r_3 IN rel_1
(r_1.compl(A) = r_2.compl(A) AND
$r_2.B = r_3.A$)].

where compl(A) is the complement of A in the attribute set of "rel_1". As the definition indicates, 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),
     ename)
```

As opposed to a relational calculus expression, which describes the relation resulting from a query by means of its properties, a relational algebra expression defines an algorithm for the construction of the resulting relation. 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; optimization opportunities may become 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 Codd [1972] 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 by Klug [1982a].

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

Nodes in *object graphs* represent objects such as (relation) variables and constants. Edges describe predicates that these objects are to fulfill [Bernstein and Chiu 1981; Palermo 1972; Youssefi and Wong 1979]. Object graphs contain the properties of the query result and are therefore closely related to the relational calculus. *Operator graphs* describe an operator-controlled data flow by representing operators as nodes that are connected by edges indicating the direction of data movement. In Smith and Chang [1975] and Yao [1979], operator graphs have been used for the representation of algebra expressions. Figures 1 and 2 give one example, respectively, for an object graph and an operator graph.

Query graphs have many attractive properties. The visual presentation of a query contributes to an easier understanding of its structural characteristics. In addition, graph theory offers a number of results useful for the automatic analysis of graphs, for example, discovery of cycles and tree property. Finally, an important advantage of query graphs is that they can be easily augmented with additional information. For example, the augmentation of graphs with details of the physical data organization of a database has been proposed by Rosenthal and Reiner [1982].

2.4 Tableaus

Tableaus as defined by Aho et al. [1979a, 1979b, 1979c] are tabular notations for a

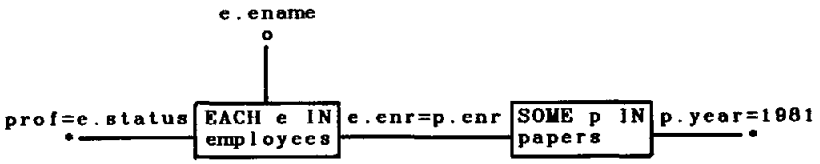

Figure 1. An object graph representing the example query.

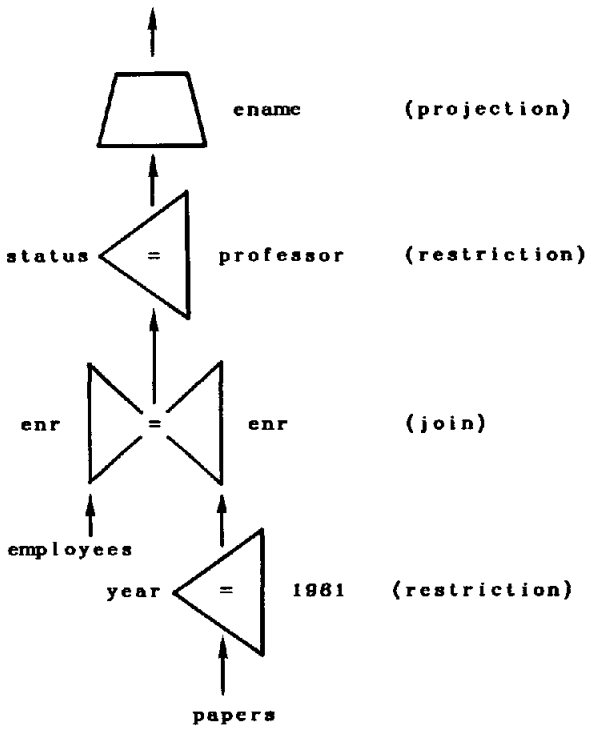

Figure 2. An operator graph representing the example query.

subset of relational calculus queries, characterized by containing only AND-connected terms and no universal quantifiers. Thus tableau queries are a particular kind of *conjunctive queries* [Chandra and Merlin 1977; Rosenkrantz and Hunt 1980].

Tableaus are specialized matrices, the columns of which correspond to the attributes of the underlying database schema. The first row of the matrix, 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 (corresponding to free variables), nondistinguished variables (corresponding to existentially quantified variables), constants, blanks, and tags (indicating the range relation).

Figure 3 illustrates the construction of a tableau representing the query of Example 2.1. It starts with tableaus for single relations and proceeds by combining these tableaus into new tableaus for larger and larger subexpressions. Distinguished variables are denoted by a's; nondistinguished ones are denoted by b's.

Expressions containing disjunction (set union) and negation (set difference) can be represented by sets of tableaus [Sagiv and Yannakakis 1980]. Klug [1983] and Johnson and Klug [1983] use sets of tableaus

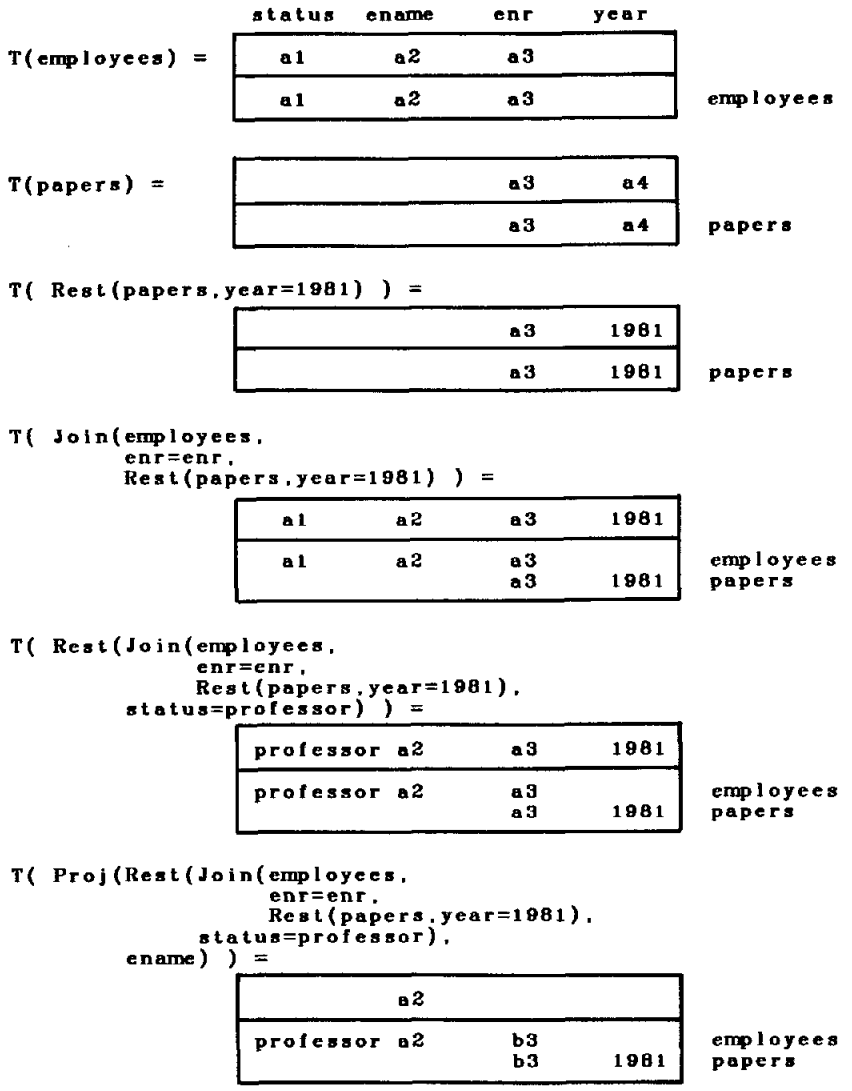

Figure 3. Stepwise construction of a tableau T representing the query of Example 2.1.

for representing general conjunctive queries. The specific value of tableaus with respect to query optimization is discussed in Section 3.2.

3. QUERY TRANSFORMATION

We have seen that queries can be expressed in a number of different representation forms. Additionally, a number of semantically equivalent expressions may exist for each query, even within a given language. The transformation of a given expression into an equivalent one by means of well-defined rules is the subject of this section. The goals of query transformation are threefold: (1) the construction of a standardized starting point for query optimization (*standardization*), (2) the elimination of redundancy (*simplification*), and (3) the construction of expressions that are improved 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 [Jarke and Schmidt 1981; Kim 1982; Palermo 1972; Wong and Youssefi 1976]. In the following, we present two normal forms for the relational calculus together with the rules to be obeyed by a 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

$$\text{SOME/ALL } r_1 \text{ IN rel}_1 \cdots \text{SOME/ALL } r_n \text{ IN rel}_n(M),$$

where M is a quantifier-free predicate. M is called the *matrix* and can also be standardized. A matrix consisting of a disjunction of conjunctions (of terms A_{ij}), such as

$$(A_{11} \text{ AND } \cdots \text{ AND } A_{1n}) \text{ OR } \cdots \text{ OR } (A_{m1} \text{ AND } \cdots \text{ AND } A_{mn}),$$

is said to be in *disjunctive normal form*, and a matrix consisting of a conjunction of disjunctions, such as

$$(A_{11} \text{ OR } \cdots \text{ OR } A_{1n}) \text{ AND } \cdots \text{ AND } (A_{m1} \text{ OR } \cdots \text{ OR } A_{mn}),$$

is in *conjunctive normal form.*

The prenex normal form combined with the normal forms for the matrix yields two normal forms for relational calculus expressions: *disjunctive prenex normal form* (DPNF) and *conjunctive prenex normal form* (CPNF). The use of DPNF is motivated by the goal to optimize and evaluate independent query components separately [Bernstein et al. 1981]. The CPNF has proved useful for the decomposition of queries [Wong and Youssefi 1976] and for data-dependent amelioration (e.g., testing the most restrictive disjunction first).

Queries in CPNF can be transformed further into a quantifier-free form popular in artificial intelligence theorem-proving applications, the so-called *clausal form* [Nilsson 1982]. Logic-based database languages such as Prolog [Kowalski 1981] are based on clausal form. Since clausal form has been rarely used in query optimization (see Grant and Minker [1981], Jarke et al. [1984], and Warren [1981] for exceptions), a detailed description is omitted at this point.

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 1. Normalization of the matrix is rather straightforward and can be achieved by using DeMorgan's rules, the distributive rules, and the rule of double negation (see Table 2).

The distinction between empty and nonempty range relations in rules Q2 and Q3 of Table 1 results from the variability of relations over time and the many sortedness of the relational calculus [Jarke and Schmidt 1982]. 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:

O1: SOME r IN rel(pred)
$\Leftrightarrow$ SOME r((r IN rel) AND pred),

O2: ALL r IN rel(pred)
$\Leftrightarrow$ ALL r((r IN rel) $\Rightarrow$ 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 normalization of an arbitrary relational calculus expression at compile time must preserve information about the original range definition of variables so that run-time modifications according to rules Q2b and Q3b can be performed when necessary.

3.2 Simplification

We have already seen that there might be several semantically equivalent expressions representing one and the same query. One source of differences between any two equivalent expressions is their degree of redundancy [Hall 1976; Stroet and Engmann 1979]. A straightforward evaluation of a redundant expression would lead to

Table 1. Transformation Rules for Quantified Expressions

```
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)))
```

the execution of a set of operations, some of which are superfluous. Therefore query optimization aims at the elimination of redundancy by means of transforming a redundant expression into an equivalent nonredundant one.

A redundant expression can be simplified by applying the transformation rules M4a to M4j, which consider *idempotency* (see Table 2). The application of these rules is complicated by the fact that idempotency can occur at any level in the expression, owing to the presence of *common subexpressions*, that is, subexpressions that occur more than once in the expression representing the query. Thus, in order to simplify an expression such as

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

to

[EACH e IN employees: e.ename = 'Smith']

by means of rules M4d and M4g, the subexpressions

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

and

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

Table 2. Transformation Rules for General Expressions

```
M1: Commutative rules
 a) A OR B              <==> B OR A
 b) A AND B             <==> B AND A
M2: Associative rules
 a) (A OR B) OR C       <==> A OR (B OR C)
 b) (A AND B) AND C     <==> A AND (B AND C)
M3: Distributive rules
 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 rules
 a) A OR A              <==> A
 b) A AND A             <==> A
 c) A OR NOT(A)         <==> TRUE
 d) A AND NOT(A)        <==> FALSE
 e) A AND (A OR B)      <==> A
 f) A OR (A AND B)      <==> A
 g) A OR FALSE          <==> A
 h) A AND TRUE          <==> A
 i) A OR TRUE           <==> TRUE
 j) A AND FALSE         <==> FALSE
M5: De Morgan's rules
 a) NOT (A AND B)       <==> NOT (A) OR NOT (B)
 b) NOT (A OR B)        <==> NOT (A) AND NOT (B)
M6: Double negation rule
    NOT (NOT (A))       <==> A
```

must first be recognized as being equivalent. Algorithms are given by Downey et al. [1980] and Hall [1974, 1976]. 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. Expressions that are bound to *empty relations* can also be simplified. Transformation rules for their simplification are given in Table 3. (Note that these rules can be applied only at run time.)

Terms as defined in Section 2.1 serve as atomic predicates in the relational calculus. However, these terms can be simplified or removed if the semantics of the comparison operators are explicitly taken into account. An important application is the so-called *constant propagation*, which uses transitivity laws, such as

$$r.A \text{ op } s.B \text{ AND } s.B = \text{const} \Rightarrow r.A \text{ op const},$$

Table 3. Transformation Rules for Expressions with Empty Relations

```
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
```

to reduce the number of dyadic terms in a query. Algorithms that minimize the number of rows in tableaus as introduced in Section 2.4 systematically exploit such simplification rules for conjunctive queries [Aho et al. 1979a; Sagiv 1981, 1983]. 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.

Sagiv and Yannakakis [1980] extend 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.

Information about semantic integrity constraints can be exploited for "semantic" query simplification [Aho et al. 1979a, 1979b, 1979c; Jarke et al. 1984; Johnson and Klug 1982; Ott and Horlaender 1982; Rosenthal and Reiner 1984]. As a simple example, consider the case of key constraints. If r and r' are (free or existentially quantified) variables ranging over the same relation "rel", equijoin terms of the form "r.key = r'.key" are superfluous in the sense that the term and one of the variables, say r', can be deleted, followed by the substitution of r' by r in any term referring to r'. This type of simplification is most relevant in the context of view processing, in which the reduction of user queries addressing views to system queries addressing stored relations may introduce substantial redundancy. Its application range can be extended by considering additional constraints implied by the query structure [Klug 1980; Koch et al. 1981].

A final opportunity for simplification occurs when one or more matrix conjunctions of the standardized query can be shown to be unsatisfiable [Eswaran et al. 1976; Klug 1983; Munz et al. 1979; Ozsoyoglu and Yu 1980]. For example, consider the expression

$$r.A \geq s.B \text{ AND } s.B > t.C \text{ AND } t.C \geq r.A,$$

which implies the contradiction, $r.a > r.a$, and can therefore be replaced by the Boolean value, false. Satisfiability can be efficiently decided at compile time for a conjunction of terms with comparison operators ($=, <, \leq, >, \geq$) [Rosenkrantz and Hunt 1980], but is computationally intractable when the nonequality comparison operator is allowed.

3.3 Amelioration

Query simplification does not necessarily produce a unique expression. Other nonredundant expressions may exist that 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, for example, the size of intermediate results and the number of relation elements accessed. Below, a number of query transformation heuristics are presented that, 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 [Hall

1976; Smith and Chang 1975]. The corresponding transformation rules are

A1: Proj(··· Proj(Proj(rel, A_1) A_2), ..., A_n)
⇔
Proj(rel, A_n),

A2: Rest(··· (Rest(Rest(rel, $pred_1$), $pred_2$), ..., $pred_n$)
⇔
Rest(rel, $pred_1$ AND $pred_2$ AND ··· AND $pred_n$).

The combination of intrarelational operations results in 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.

Minimization of the size of intermediate results to be constructed, stored, and retrieved is the goal of a number of ameliorating transformations. An 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 operations as early as possible [Smith and Chang 1975]. 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 Example 3.1.

Example 3.1. A nested expression equivalent to the expression in Example 2.1.

```
[⟨e.ename⟩ 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 forms a special case of *query detachment* as introduced by Wong and Youssefi [1976]. A subexpression that overlaps with the rest of the expression on a single variable is detached and forms an inner nesting. Detachment is performed recursively at any nesting level until the expression cannot be reduced further. Experiments reported by Youssefi and Wong [1979] have shown this heuristic to be very strong. Example 3.2 demonstrates the detachment of a subexpression in a complex expression.

Example 3.2. 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 is

```
[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 is

```
[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 4.

Note that the resulting nested expression is irreducible [Goodman and Shmueli 1980]; that is, it cannot be separated into two subexpressions overlapping on a single variable. In other words, the nested expression contains a cycle (see Figure 4).

The importance of the distinction between cyclic and acyclic (treelike) expressions for query processing is discussed further in Section 4.3. At this point, we mention only that there are cycles that can be transformed into equivalent acyclic

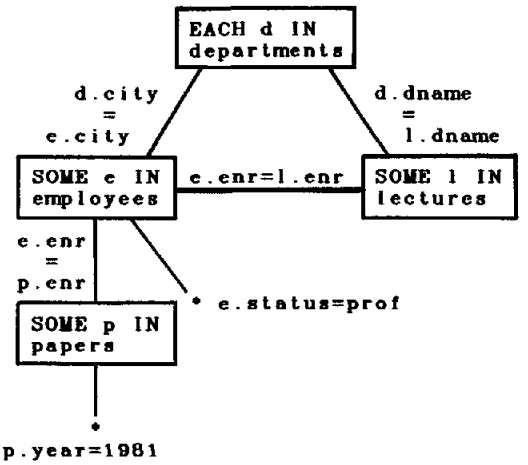

Figure 4. Object graph for Example 3.2.

query graphs. Such cycles include those that (1) are introduced by transitivity [Bernstein and Chiu 1981; Yu and Ozsoyoglu 1979], (2) contain certain combinations of inequality join term edges [Bernstein and Goodman 1981b; Ozsoyoglu and Yu 1980], (3) are "closed" by universally quantified variables [Jarke and Koch 1983], and (4) contain variables that can be decomposed by use of functional dependencies [Kambayashi and Yoshikawa 1983].

The concepts of *extended range expressions* [Jarke and Schmidt 1982] and *range nesting* [Jarke and Koch 1983] provide a generalization of query detachment in that they also consider 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: pred_1 AND pred_2]
$\Leftrightarrow$
[EACH r IN [EACH r IN rel: pred_1]: pred_2],

A4: SOME r IN rel(pred_1 AND pred_2)
$\Leftrightarrow$
SOME r IN [EACH r IN rel: pred_1] (pred_2),

A5: ALL r IN rel: (NOT(pred_1) OR (pred_2)
$\Leftrightarrow$
ALL r IN [EACH r IN rel: pred_1] (pred_2).

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.

The ameliorating transformations presented thus 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 that have not yet been considered are the integrity constraints that complement the structural schema definition in many database systems and the actual data stored in the database.

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. They therefore can be added to the selection expression of any query without changing its truth value. There are a few approaches exploiting this observation for amelioration

rather than simplification (which was mentioned in the previous subsection). They have been called knowledge-based [Hammer and Zdonik 1980] or *semantic query processing* [King 1979, 1981].

Assume, for example, that an integrity constraint says: "We only hire professors who have published at least one paper per year." In this case, the evaluation of Example 2.1 (asking for professors with papers published 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 in order to make it more tractable. Consider the constraint: "We only hire local professors." In this case, the term "d.city = e.city" in Example 3.2 can be omitted. The remaining query no longer contains a cycle in its object graph.

The success of semantic query processing depends largely on the development of efficient heuristics for choosing among the many transformations made possible by the addition of any combination of integrity constraints to the query. In King [1981] and Xu [1983], artificial intelligence type rules are used to make this decision for a special class of relational databases.

Yao [1979] points out that cases exist in which 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, the run-time support must also be equipped with query transformation facilities. Furthermore, if heuristics do not yield satisfactory results, simultaneous optimization is required on the physical and the logical level. Before turning to such integrated approaches, however, the physical evaluation of query components must be described.

4. QUERY EVALUATION

In this section we present methods for the evaluation of query components of varying complexity, such as one-variable expressions, two-variable expressions, and multivariable 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 to test whether it satisfies each term of the expression. Since this approach is very costly in the presence of large relations and complex expressions, various techniques have been used to reduce the number of element accesses and 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 those of exhaustive sequential search. 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, or binary search can be performed. This has proved useful for the evaluation of range queries (i.e., expressions that define an interval of attribute values [Bolour 1981; Davis and Winslow 1982]). Another alternative, hashing, provides fast direct access without necessarily preserving order.

Direct and ordered access can be provided by *indexes*, possibly combined with multilist structures [Welch and Graham 1976; Yang 1977]. Conceptually speaking, an index is a binary relation that associates attribute values with references to relation elements, usually called tuple identifiers (TIDs). We distinguish *one-dimensional indexes*, which support access via a single relation attribute, from *multidimensional indexes*, which support access via a combination of attributes. One-dimensional indexes are usually implemented by ISAM [IBM 1966] or B-tree [Bayer and McCreight 1972] structures. An overview of multidimensional index structures is given

by Bentley and Friedman [1979]. Some examples include the work by Shneiderman [1977] on combined indexes, Nievergelt et al. [1984] on grid files, and Gardarin et al. [1984] on predicate trees. Although access paths are usually invisible to the user, efforts have been reported to develop high-level language representations directly available to those database programmers who insist on extreme efficiency. Such language constructs range from TIDs [Jarke and Schmidt 1981; van de Riet et al. 1981] to abstract representations of complete access paths [Mall et al. 1984; Schmidt 1984; Tsichritzis 1976].

The number of tests applied to an accessed relation element during expression evaluation can be reduced by means of run-time transformations of the expression. The optimization of a special class of expressions, *Boolean expressions*, has been a research topic in compiler construction for a long time [Gries 1971]. 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 purpose of code optimization for 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 as "false". If the same idea is applied to the evaluation of one-variable expressions in query languages [Gudes and Reiter 1973; Liu 1976], queries can be simplified at run time.

Another approach designed to improve evaluation efficiency is that of changing the order in which individual expression components are evaluated. Several algorithms are known to lead to optimal evaluation sequences in certain situations; some assume a priori probabilities for attribute values [Hanani 1977], whereas others work without such assumptions [Breitbart and Reiter 1975]. Warren [1981] applies a similar technique for optimizing database programs expressed in logic.

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, which restrict single variables independently of each other, and dyadic terms, which establish the link between both variables. In this section we first describe the basic methods for the evaluation of a single dyadic term, corresponding to the join operator in Section 2.2, and then strategies for the evaluation of arbitrary two-variable expressions.

Approaches to the implementation of the join operation can be classified into order-dependent and order-independent strategies [Todd 1974]. A simple method that is independent of the order of element access is the so-called *nested iteration* method [Pecherer 1975, 1976; Selinger et al. 1979] in which every pair of relation elements is accessed, and concatenated if the join condition is satisfied. A sketch of the algorithm follows:

```
FOR i := 1 TO N1 DO
  read ith element of rel1;
  FOR j := 1 TO N2 DO
    read jth element of rel2;
    form the join according to the join condition;
  END;
END;
```

Let $N_1(N_2)$ be the number of elements of the relation read in the outer (inner) loop. $N_1 + N_1 * N_2$ 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 index on the join attribute(s) of "rel_2." Instead of scanning "rel_2" sequentially for each element of "rel_1," the matching "rel_2" elements are retrieved directly [Griffeth 1978; Klug

1982b]. Thus only $N_1 + N_1 * N_2 * j_{12}$ accesses are required, where j_{12} is a join selectivity factor describing the reduction of the Cartesian product of "rel_1" and "rel_2" by the join condition.

The *nested block method* [Kim 1980] adapts the nested iteration method to a paged-memory environment. It assumes that a main memory buffer will hold one or more pages of both relations, where each page contains a set of records.

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 bo $P_1 + (P_1/B_1) * P_2$, where $P_1(P_2)$ is the number of pages occupied by the outer (inner) relation and B_1 is the number of pages of the outer relation held in the main memory buffer. The formula demonstrates that it is always preferable to read the smaller relation in the outer loop (i.e., make $P_1 < P_2$). Note that only $P_1 + P_2$ accesses are necessary if one of the relations can be kept entirely in the main memory buffer.

The *merge method* [Blasgen and Eswaran 1977; Selinger et al. 1979] is based on the 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 $N_1 + N_2 + S_1 + S_2$ secondary storage accesses are required, where S_1 and S_2 denote the number of secondary storage accesses necessary to sort the relations. If the two relations are already sorted by the same criterion, the merge method appears to be the most efficient one for evaluating a dyadic term [Merrett 1981; Merrett et al. 1981]. Exceptions occur when one of the two relations is small enough to fit in main memory (the nested block method is preferable) or one relation is much larger than the other and indexes are available (nested iteration with indexes is better).

Methods for the evaluation of *arbitrary two-variable expressions* are created on the basis of strategies for one-variable expressions and algorithms for the computation of dyadic terms. They differ in the way they make use of or even create access paths, such as indexes and sorting, and the order in which the terms are processed. One such method is illustrated in Figure 5. It 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 finally projected on the target attribute.

Blasgen and Eswaran [1976], Niebuhr and Smith [1976], Yao and DeJong [1978], and Yao [1979] present various other algorithms and compare them systematically with respect to their efficiency. Their results demonstrate that often no a priori best algorithm exists. The optimizer must either rely on heuristics or perform an expensive cost comparison of many alternatives for each query.

An important case of two-variable expressions is a join in which one of the participating variables (the "inner" one) is existentially or universally quantified. The result of such an expression contains only elements of one relation. Furthermore, access to the other relation needs to establish only whether, for a given value of the "outer" variable, the join condition is satisfied with any (respectively all) elements of the range relation of the "inner" variable. These elements themselves are of no interest. This means that for the evaluation of quantified queries intermediate results can be represented in a compressed fashion [Dayal 1983a; Jarke and Schmidt 1981]. If the two-variable expression contains just one join term with a comparison operator of $<$, $\leq$, $>$, or $\geq$, only one attribute value (the minimum or maximum attribute value appearing in the inner relation) is required. That is, the quantified two-variable expression can be converted into a monadic term [Jarke and Koch 1983]. Incidentally, a similar use of aggregate functions (maximum or minimum) has also been proposed in the context of integrity maintenance [Bernstein et al. 1980].

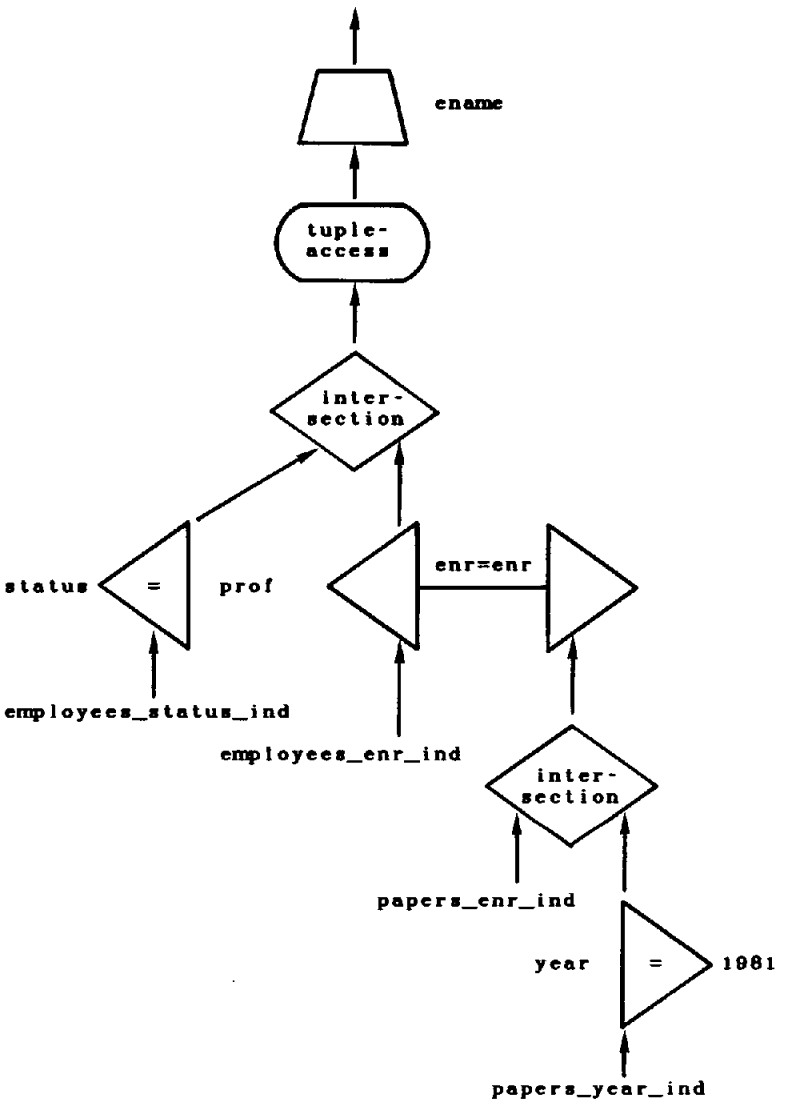

Figure 5. Operator graph illustrating the evaluation of the query of Example 2.1. The existence of various indexes is assumed.

4.3 Multivariable Expressions

Stategies for the evaluation of multivariable expressions are the largest building blocks for a general query-processing system. Two basic approaches exist, which are referred to as parallel processing and stepwise reduction.

The *parallel processing* of query components serves to avoid repeated access to the same data. Repeated access to the same data can be avoided by simultaneous evaluation of multiple-query components. In Palermo [1972], all monadic terms associated with a variable are completely evaluated and all dyadic terms in which the same variable appears are partially processed concurrently with scanning of the range relation of the variable. Even AND-connections existing among the terms can be evaluated in parallel, which further reduces the size of intermediate results [Jarke and

Schmidt 1982]. A similar approach on a higher level has been described by Klug [1982b] in which aggregate functions and complex subqueries are computed in parallel. Scheduling strategies for the parallel processing of query components are discussed by Schmidt [1979].

Pipelining operations that can work on the partial output of preceding operations is another technique that exploits parallel processing opportunities [Smith and Chang 1975; Yao 1979]. For example, restriction and projection can be pipelined so that only a relatively small buffer for data exchange is required rather than the creation and subsequent reading of a possibly large temporary relation.

Aspects of simultaneous evaluation and pipelining are combined in the so-called "feedback" method [Clausen 1980; Rothnie 1974, 1975], which uses 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 occurring in the join term. For example, consider the expression

$$[\text{EACH } r_1 \text{ IN rel}_1: \text{ ALL } r_2 \text{ IN rel}_2 \ (r_1.A \text{ op } r_2.B)].$$

Assume that the join term is evaluated by nested iteration. While testing some element, r_1, it is found that the term "$r_1.A$ op $r_2.B$" evaluates to false for a certain r_2 with $r_2.B = c_1$. Because of the universal quantification of r_2, r_1 is rejected, and an elimination filter can be added to the selection expression

$$[\text{EACH } r_1 \text{ IN rel}_1: \text{ NOT } (r_1.A \text{ op } c_1) \text{ AND ALL } r_2 \text{ IN rel}_2 \ (r_1.A \text{ op } r_2.B)]$$

because the same r_2 would cause the rejection of all elements r_1 of "rel$_1$" that do not satisfy the first term. On the other hand, if r_1 with $r_1.A = c_2$ passes the test, a true filter can be added to the selection predicate:

$$[\text{EACH } r_1 \text{ IN rel}_1: r_1.A \text{ op } c_2 \text{ OR NOT } (r_1.A \text{ op } c_1) \text{ AND } \cdots].$$

Both filters can be updated subsequently to sharpen the constraints.

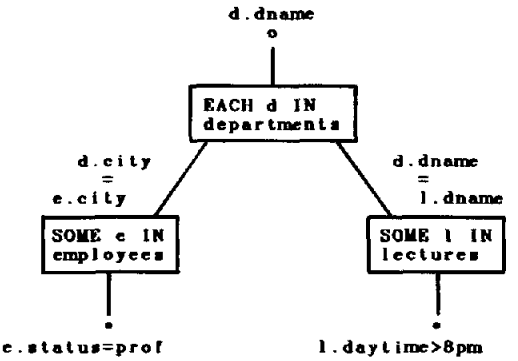

Figure 6. Object graph for Example 4.1.

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

Example 4.1. Figure 6 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 > 8 pm
                        AND
                        l.dname = d.dname)]
```

Expressions like the one in Example 4.1 are called *tree expressions* [Goodman and Shmueli 1982; Shmueli 1981], since their associated query graph is a tree. The standard approach for 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 has been shown in the introductory example (Section 1.2, Strategy 2), this method performs rather poorly. It is even more problematic in a distributed environment where each relation resides on a different site, as entire relations must be transmitted from site to site. Moreover, the relation at the target site is temporarily expanded through the formation of the join, although the final result is only a horizontal and vertical subset of it.

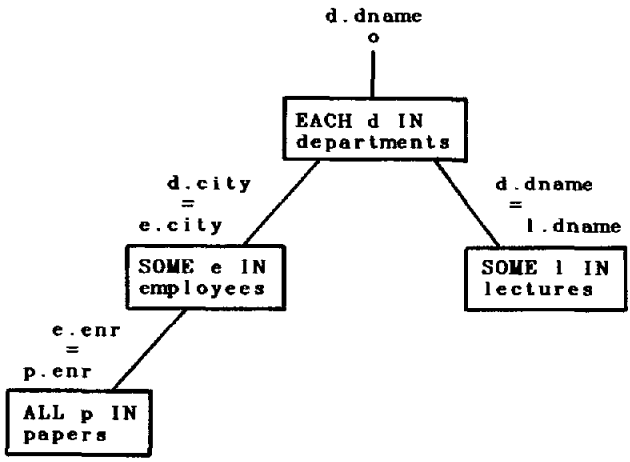

Figure 7. Object graph for Example 4.2.

In Bernstein and Chiu [1981], Bernstein et al. [1981], and Bernstein and Goodman [1981a, 1981b, 1981c], the *stepwise reduction* of tree expressions (with free and existentially quantified variables) has been introduced. This method often outperforms the simple approach above in a decentralized as well as in a centralized setting. It is based on a modified join operation, and uses the so-called semijoin operator.

The *semijoin* of a relation "rel_1" by a relation "rel_2" equals the join of these relations projected back onto the attributes of relation "rel_1":

$$\text{semijoin}(rel_1, A \text{ op } B, rel_2) = \text{proj}(\text{join}(rel_1, A \text{ op } B, rel_2), \text{attributes}(rel_1)).$$

The operation thus forms "half of a join." The main advantages of semijoin over join are as follows: (1) its evaluation only requires the transmission of value lists of the joining attributes instead of that of an entire relation, and (2) it has a "reductive" effect since the result of semijoin (rel_1, A op B, rel_2) is always a subset of rel_2, whereas a join may produce a Cartesian product in the worst case. In terms of relational calculus, a semijoin corresponds to a two-variable expression, where one variable is existentially quantified, as discussed in Section 4.2.

The simplest evaluation of a tree expression by means of stepwise reduction can be described 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 one free and $n - 1$ existentially quantified variables can be completely processed by $n - 1$ semijoins.

If all variables are free, an additional semijoin sequence in reverse order must be performed. In a centralized setting, this kind of semijoin strategy is inferior to a merge join operation combined with parallel quantifier evaluation. Even in a distributed system, the simple bottom-up/top-down sequence is often less efficient than a middle-out sequence that begins with semijoins that achieve very strong reduction [Chiu and Ho 1980; Chiu et al. 1981].

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 possible only 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 7 representing 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)]
```

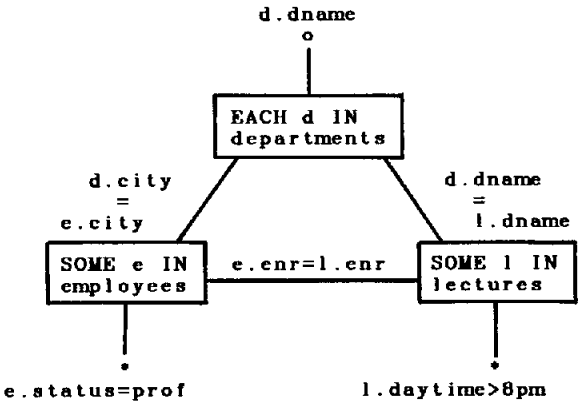

Figure 8. A cyclic calculus expression and its corresponding object graph.

Figure 9. Some possible database state.

departments

dname	city	street address
d1	c1	s1
d2	c2	s2

employees

enr	ename	status	city
e1	en1	st1	c1
e2	en2	st2	c2

lectures

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

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 cases where rules Q1 through Q4 of Table 1 apply. Expressions containing only one type of quantifier allow the sequence of variables to be interchanged arbitrarily, according to transformation rules Q5 and Q6. Jarke and Koch [1983] describe a directed, so-called "quant graph" that supports the application of these rules.

Cyclic expressions are the complement of tree expressions with respect to the entire set of expressions. Although we quoted some benign exceptions in Section 3.3, cyclic expressions in general cannot be fully reduced by means of semijoins [Bernstein and Goodman 1981a, 1981b; Goodman and Shmueli 1982].

Example 4.3. Consider the query: "names of departments that offer lectures after 8 pm 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 8.

If the database is in the state shown in Figure 9, no sequence of semijoins (corresponding to edges of the query graph of Figure 8) will produce the correct result (the empty relation). The reason is that the semijoin technique only considers one edge at a time, and thus loses restrictive conditions introduced through the feedback effect of the cycle:

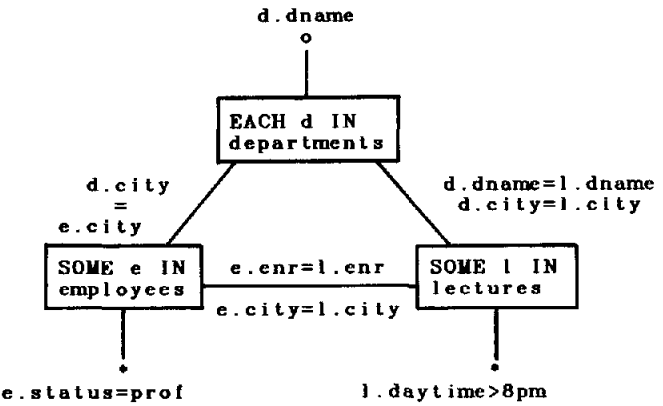

Figure 10. Augmented object graph for Example 4.3.

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

In Kambayashi et al. [1982] a proposal is made that is intended 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 edges of the graph. Figure 10 demonstrates the technique applied to the cyclic expression of 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 query, a special form of tree query. Note that addition of the new terms corresponds basically to addition of the city attribute to the schema of the lectures relation (initialized with null values).

The query tree then is processed by stepwise reduction, executing a generalized semijoin for each edge while taking into account the newly introduced attributes. The number of data transfers are reduced by means of specialized compression techniques.

Methods for the efficient implementation of operations, such as the ones presented in this section, are candidates for hardware components in specialized database machines. However, such components often allow parallelism and therefore require somewhat different join and semijoin algorithms [Bitton et al. 1983; Maekawa 1982; Missikoff and Scholl 1983; Valduriez 1982; Valduriez and Gardarin 1984]. A brief survey of hardware approaches to query optimization is presented in Section 6.3.

5. ACCESS PLANS

In the previous section we dealt with techniques for the efficient evaluation of query components that can be used as building blocks of a general query evaluation algorithm. The final step of our query evaluation framework requires the combination of these blocks into an efficient evaluation procedure for an arbitrary standardized, simplified, and ameliorated expression. Unfortunately, work in this area is still incomplete.

The inputs of such a procedure are a logically preprocessed query (as described in Section 3), the existing storage structures and access paths, and a cost model. The output is an optimal (or at least heuristically "good") access plan. The procedure consists of the following steps.

(1) Generate all reasonable logical access plans for evaluating the query. A logical access plan describes a sequence of operations or of intermediate results leading from existing relations to the final result of a query.

(2) Augment the logical access plans by details of the physical representation of data (sort orders, existence of physical access paths, statistical information).
(3) Choose the cheapest access plan by applying a model of access and processing costs.

In this section we review the generation of access plans, cost models for their evaluation, and the problem of selecting the cheapest plan. The quality of the final solution plan is strongly influenced by the existing storage structures and access paths, which usually cannot be optimized for a single ad hoc query. In Section 5.4 we therefore briefly consider the simultaneous optimization of multiple queries.

5.1 Generation of Access Plans

Access plans describe sequences of operations (represented by operator graphs) or intermediate results (object graphs) leading from the existing data structures to a query result. The 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 exemplified by Smith and Chang [1975] and Yao [1979]. Smith and Chang use a rigid set of "automatic programming" query transformation rules similar to the ones discussed in Section 3. Their procedure generates exactly one access plan, which need not be optimal. Yao's method generates all nondominated 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 seek a compromise between heuristic selection and detailed generation of alternative access plans. For example, System R [Chamberlin et al. 1981; Selinger et al. 1979] 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. Kim [1982] notes that this concept places too much emphasis on the user-specified block structure of the query and therefore introduces query standardization steps into SQL query processing.

A similar compromise was chosen in INGRES [Youssefi and Wong 1979]. 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 comprehensive procedure for generating access plans to solve conjunctive queries without universal quantifiers and aggregates has been proposed by Rosenthal and Reiner [1982]. An expanded object graph representation is used for modeling evaluation strategies that exploit auxiliary direct access structures. To avoid the generation of an excessive number of strategies to be generated, the generation of access plans is interleaved with the selection step: Strategies known to be infeasible or dominated by other procedures are not created.

5.2 Cost Analysis of Access Plans

The selection of physical access plans is determined by heuristic rules or is based on a cost model of storage structures and access operations [Merrett 1977]. In this section, cost models and their integration into optimization procedures are reviewed.

Whereas a few researchers consider working storage requirements [Kim 1982; Lang et al. 1977; Sacco and Schkolnick 1982] or CPU costs [Gotlieb 1975; Selinger et al. 1979], 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, the access structures used, and the size of main memory buffers.

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 by using information about the original data structures and the selectivity of the operations already performed on them. Although there

is much ongoing work, no generally accepted formulas for *estimating the size of intermediate results* have evolved thus far. This results in part from the fact that the trade-off between the amount of information used and the accuracy of the result is not very well understood.

In general, the more restrictive the assumptions about the data, the fewer are the parameters needed to compute the size of the results of operations. For example, in Demolombe [1980] a recursive procedure for estimating the size of a quantifier-free calculus expression is described, for which five types of parameters must be known if fairly detailed database statistics are available. Under more restrictive assumptions, however, only three of them are needed. The size estimates given by Selinger et al. [1979] use only information already existing in the database but make many assumptions about data and queries [Astrahan et al. 1980]. At the other extreme, Yao [1979] assumes (implicitly) that detailed selectivity data are known; no statement is made as to how these data are obtained. (See, however, Yao [1977b] for an access cost model.)

More recently researchers recognized the need to carefully state and critically review all underlying assumptions about database characteristics to generate formally valid parameter systems that allow one to

(1) compute a size estimate for any feasible operation, and
(2) compute parameter values for intermediate results required for further operations.

Such techniques view the database state at run time as the result of a random process that generates relation elements from the Cartesian product of the attribute domains, governed by some probability distribution (usually assumed to be uniform) and by general laws (e.g., functional dependencies) of the database schema [Gelenbe and Gardy 1982; Richard 1981]. From these assumptions, parameters are derived whose values must be known in order to compute the size of any intermediate result of complex operations. For example, Richard [1981] demonstrates that it is sufficient to know the size of all projections in the database if the attribute value distributions are uniform and independent both within an attribute and between attributes of the same domain.

Christodoulakis [1981, 1983] and Montgomery et al. [1983] have critically reviewed such assumptions and have proved that they lead to a bias against direct-access structures in selection plans. However, no practical formulas with more general assumptions, but without excessive data requirements, have yet been published.

The final cost measure is the *number of secondary storage accesses*, not the sizes of intermediate results. A large number of researchers estimate the relationship between the two figures [Cardenas 1975; Chan and Niamir 1982; Cheung 1982b; Luk 1983; Whang et al. 1983; Yao 1977a; Yu et al. 1978]. In essence, it depends on the physical storage structures involved and the proportion of elements to be accessed.

Assume first that all elements of an operand of size N have to be accessed. The optimal number of secondary storage accesses would then be N/B, where B is the blocking factor of the operand. This can be achieved only if the elements are stored densely and it is clear from the beginning on which physical records the elements reside. As a counterexample, the so-called "segment scan" of System R requires access to a superset of the necessary pages to find all elements of a relation [Selinger et al. 1979]. If it is necessary to read the elements in some predetermined sequence, they 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 required to retrieve n of the N elements will depend on the clustering of elements in physical blocks. Most of the models cited above assume random placement of records on pages, which in some sense describes a worst case [Christodoulakis 1981]. Optimal clustering can reduce the number of pages to be accessed to n/B.

In conclusion, the traditional assumptions about value distributions and element placements tend to overestimate costs and thus to bias cost estimates against the use

of direct-access structures. On the other hand, 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.

5.3 Selection of Access Plans

How are the cost estimates used in query optimization? As mentioned in Section 5.2, 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" [Youssefi and Wong 1979]. Experiments indicate that combinatorial analysis can improve database performance considerably [Epstein and Stonebraker 1980].

There are two ways to utilize cost estimates in the selection process. First, the costs of each alternative access plan can be determined *completely* [Blasgen and Eswaran 1976; Yao 1979]. This approach has the advantage of covering techniques like parallelism 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. This approach allows whole families of strategies with common parts to be evaluated in parallel, which considerably reduces the optimization costs. For example, the method proposed by Rosenthal and Reiner [1982] retains only the cost minimal way to obtain each intermediate result, and discards any other method as soon as its nonoptimality is detected.

An extension of this second approach is a *dynamic* query optimization procedure, which derives from the observation that, at each moment, only the next operation to be performed has to be decided. To guarantee overall optimality, only the consequences of this decision for the remainder of the evaluation 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. The dynamic method has two drawbacks: its cost and the danger of getting stuck in local optima if no look ahead is applied. However, if used carefully, the method can reduce the evaluation costs for queries, in which the sizes of actual intermediate results differ from the expected sizes.

5.4 Support for Multiple Queries

All of the query evaluation procedures considered thus far concentrate on optimizing the evaluation of a single query. Chesnais et al. [1983] have also investigated the performance effect of multiple users accessing a database in parallel. However, query optimization strategies can even go beyond simple parallelism by *sharing* the execution costs of common operations among queries. Additionally, a strategy that optimizes the evaluation of multiple queries simultaneously can consider *investments* in additional access paths, the creation of which would not be cost effective for a single query. The few existing approaches to multiple-query optimization can be classified in three groups, according to the time scope for which decisions are made: (1) simultaneous optimization of batched queries, (2) index selection, and (3) physical database design.

A set of queries submitted by one or more users at approximately the same time can be *batched* for more efficient evaluation [Shneiderman and Goodman 1976]. The techniques for batched evaluation are similar to those described in Section 4.3 for multivariable expressions. First, results of common subexpressions can be shared among queries [Grant and Minker 1981; Jarke 1984], and subexpressions accessing the same physical data page can do so with one secondary storage access. Second, temporary physical access paths such as sorting, hashing, or indexes can be provided whose costs pay off for the batch as a whole. Finally, results of some queries can be retained for processing subsequent queries [Finkelstein 1982; Hevner and Yao 1981]. There seems to be no coherent theory in this area yet. Kim [1981, 1984] and Jarke [1984] present language constructs and preliminary architectures, and a number of

ongoing research projects are described in IEEE [1982].

Many of the examples in this paper have demonstrated the importance of using *indexes* for the performance of query evaluation algorithms. From this viewpoint, indexes can hardly hurt anywhere but are most profitable if they are very selective and support access to attributes frequently referred to in queries [Gilles and Schuster 1975]. 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 surveys [Batory 1982] and tutorial papers [March 1983; Putkonen 1979; Schkolnick 1975; Severance and Carlis 1977]. Selection and maintenance of more general indexes that support user views have been investigated by Roussopoulos [1982a, 1982b]. The use of high-level language constructs for extended view concepts has been discussed by Jarke [1984] and Schmidt [1984].

Finally, query optimization influences the *physical database design*. However, long-term query optimization is only one of many aspects of physical database design (see, e.g., Carlis et al. [1981], Carlson and Kaplan [1976], Schkolnick [1982], and Teorey and Fry [1982]). Important design strategies with an impact on query processing efficiency include the horizontal clustering of relation elements by attribute values [Salton 1978] and the vertical partitioning of attributes by frequency of combined access [Hammer and Niamir 1979].

6. NONSTANDARD QUERY OPTIMIZATION

We have described query optimization in the framework of relational calculus queries in centralized database systems. While this approach covers much of the work done in the area, some query-processing problems exceed the framework either because of a query complexity that goes beyond relational completeness or as a result of the structure of the underlying physical database. Without claiming completeness, in the following sections we briefly survey some important developments. The reader is referred to the cited literature for further details.

6.1 Higher Level Queries

Queries expressed in a relationally complete language retrieve such relation elements (or sets of elements) from a database that can be described by a predicate of the relational calculus. Whereas this method is sufficient for most transaction-oriented business applications, certain other applications may require more complex data objects or more powerful query predicates. These requirements can be characterized as language extensions that yield query languages more powerful than the relationally complete ones. Optimization techniques for such extensions are addressed in this section.

Hierarchical and network database systems support data objects that are more complex than flat records and thus contain information-bearing access paths [Astrahan and Ghosh 1974]. Relational interfaces to such systems therefore require efficient translation of relational queries to navigational access programs. (The reverse problem—program conversion from network code to relational queries—has also been studied [Katz and Wong 1982].) Several approaches have been described: interpretative, translational, and view processing.

Interpretative models (e.g., Zaniolo [1979]) directly interpret relational queries as sequences of tuple-at-a-time operations on a network database. Similarly, direct translation methods (e.g., Vassiliou and Lochovsky [1980]) frequently do not address optimization; such methods may yield quite inefficient code. Dayal and his co-workers [Dayal et al. 1981; Dayal and Goodman 1982] and Gray [1981, 1984] have developed query optimization strategies for network databases. Alternatively, one can represent network structures as relational views with particular integrity constraints [Rosenthal and Reiner 1984]. In this way, existing relational view optimization facilities can be used for compiling an efficient navigational program working on the network database.

Applications in *computer-aided design and manufacture* and *text processing* usually work on even more complex objects composed of related elements from different relations. On top of a traditional relational (or network) database, one can define structures that allow access to either the whole object or to part of it, using multiple views of the data [Johnston et al. 1983]. Another approach is an extension of the relational model by nonfirst normal form relations [Lamersdorf 1984; Schek and Pistor 1982]. Such extensions support access to substructures by specialized indexing schemes or use pattern-searching mechanisms similar to those used in information retrieval [Davis and Kunii 1982]. Dayal [1983b] investigates the related problem of answering queries in generalization hierarchies, which has to take into account the fact that data objects inherit properties of more general objects.

A relational calculus query retrieves a set of data as they come from the database, but does not support grouping and abstracting from the stored data to more complex data objects within the query language. The grouping of and summarizing over elements of the same relation by certain so-called category attributes lies in the domain of *statistical* query processing. Some query languages offer a limited set of built-in aggregate functions to support such applications. Aggregate functions can be defined as extensions to either the relational calculus [Klug 1982a] or the relational algebra [Ozsoyoglu and Ozsoyoglu 1983], and can be evaluated by using nested iteration with indexes as described in Section 4 [Klug 1982b]. However, many statistical databases are characterized by high redundancy and a large percentage of null values. Therefore the data must be compressed and stored in ways that differ from standard relational databases. An overview of these issues can be found in Shoshani [1982]. Special software [Eggers and Shoshani 1980] and hardware [Bancilhon et al. 1982; Hawthorn 1982] have been devised for processing queries on such databases. Walker [1980] discusses the use of small abstracted databases in decision support systems.

Database applications in *artificial intelligence*, especially expert systems [Nau 1983] and natural language user interfaces [Marburger and Nebel 1983; Sagalowicz 1977], require inferences to be performed over the raw data coming from the database [Buneman 1979; Chang 1979; Minker 1975, 1978]. Whereas parts of such an inference mechanism can be provided by traditional view mechanisms with some additional optimization [Jarke et al. 1984; Ott 1977; Ott and Horlaender 1982; Paige 1982], more complex requests—use of recursion, for example—must be treated in a different way.

A number of alternative architectures for coupling expert systems with DBMSs are presented by Jarke and Vassiliou [1984]. A request to the expert system usually translates to a sequence of related database calls. Optimization techniques include the combination of multiple tuple-oriented database calls into set-oriented operations [Kunifuji and Yokota 1982; Vassiliou et al. 1984], the simplification of such retrieval requests [Grishman 1978; Jarke et al. 1984; Reiter 1978], the reordering of conditions to be tested [Warren 1981], and the sharing of intermediate results [Grant and Minker 1981], which is particularly useful in executing recursive database calls [Chang 1978; Henschen and Naqvi 1984; Kellogg 1982; Minker and Nicolas 1983]. A tool often proposed in this context is the logic programming language Prolog [Kowalski 1981]. Parsaye [1983] proposes extensions to Prolog specifically designed for database and knowledge-base management.

The language extensions presented in this section can be characterized theoretically by their expressive power and by the difficulty of their evaluation. Chandra and Harel [1982a, 1982b] analyze several classes of higher level query languages, such as (in order of increasing language power and computational complexity) first-order relational calculus queries, Horn clause queries, fix-point queries, second-order queries, and general computable queries. The user of a traditional query language has to achieve the power of such high-level languages through the use of general programming language constructs in a data-

base programming language [Schmidt 1984]. The disadvantage of this solution, besides the increased programming effort, is that the responsibility for efficient implementation shifts from the database management system to the user.

6.2 Distributed Databases

In a distributed database fragments of one logical database (the database as seen by the user) reside on several physical databases, each accessed by a separate computer. Databases can be distributed for higher availability of data (through data replication [Andler et al. 1982]), improved accessibility by the most frequent users (through local storage of data [Wong 1983]), and increased execution speed of queries (through parallel processing of fragments [Su and Mikkilineni 1982; Wong and Katz 1983]). Examples of distributed database systems include Distributed INGRES [Stonebraker and Neuhold 1977], POREL [Neuhold and Biller 1977], SIRIUS [Esculier and Clorieux 1979], VDN [Munz 1979], SDD-1 [Bernstein et al. 1981], and R* [Williams et al. 1982].

The fact that data are physically dispersed and may be replicated strongly influences database design [Chen and Akoka 1980], concurrency control [Bernstein and Goodman 1981c], and query processing. Rothnie and Goodman [1977] give an overview of the major research issues.

If data are distributed, the cost of data transfer becomes a decision variable rather than a constant in the query optimization problem. Since communication costs tend to dominate local processing costs, query optimization requires a completely different objective function: the minimization of communication delay, often represented by the amount of data transmitted from one site to another.

The primary decision influencing data transfer is the selection of the site(s) where computations are performed. The general strategy is to distribute the evaluation of the query rather than collect all the data and execute the query at one site (e.g., where the query was issued). The benefits of this approach have been impressively demonstrated by Tanenbaum [1981] and Gavish and Segev [1982]. Within this general strategy, the most important tactical objectives are the maximal reduction of data to be transmitted by local preprocessing [Forker 1982], and the selection of the site(s) where the global operations are performed (e.g., Bernstein et al. [1981] and Ceri and Pelagatti [1982]). If data are replicated, there is a choice of which copy to use in order to minimize data transfer [Ullman 1982; Williams et al. 1982].

A large number of distributed query processing strategies have been devised. The choice among such strategies is influenced by several factors.

6.2.1 *Trade-Off between Communication and Local Processing Costs*

If the communication lines are relatively slow, communication costs usually dominate the costs of distributed evaluation to the extent that all other costs become negligible. Some of the more theoretical models (e.g., Apers et al. [1983], Hevner [1979], and Muthuswamy and Kerschberg [1983]) ignore local processing costs altogether. In practice [Bernstein et al. 1981; Smith et al. 1981] there will be a two-level optimization procedure that first plans the global strategy and then develops an optimal subquery evaluation plan for each local site.

As the communication speed increases, local processing costs have to be taken into account [Chu and Hurley 1982; Gouda and Dayal 1981]: No longer can an arbitrary amount of local preprocessing be justified to reduce data transfer between sites. Kerschberg et al. [1982] report experience with a two-computer network that demonstrates the relative importance of local processing. Of course, simultaneous minimization of data transfer and local processing increases the number of alternatives to be compared considerably since the problem no longer can be decomposed into a hierarchy of independent subproblems.

6.2.2 *Network Structure*

The topology of the computer network on which the distributed database is imple-

mented has an impact on the complexity of query optimization. In networks with arbitrary message routing, queuing delays at intermediate nodes play an important role [Muthuswamy and Kerschberg 1983; Tanenbaum 1981]. However, they can be considered only within a global optimization of all network traffic. To avoid this complication, many optimization algorithms (e.g., Hevner and Yao [1979]) assume a fully connected network with node-independent communication delays. Another simple structure is a star computer network [Kerschberg et al. 1982], in which a major central processor is connected to several (usually smaller) local processors.

The network structure can be homogeneous (i.e., it can consist of processors of the same type) or heterogeneous. For a homogeneous network (equal processor speed, processor-independent linear communication costs), Chu and Hurley [1982] have proved a number of theorems, the application of which restricts the search space for optimal solutions. For instance, the local preprocessing of monadic operations (restriction, projection) is shown to be globally optimal under these assumptions. Even within a homogeneous computer network, a heterogeneous distributed database may exist if the local databases work on different data models or DBMS [Chan et al. 1983; Smith et al. 1981].

6.2.3 Data Distribution Strategy

A relation can be physically partitioned horizontally (the set of tuples is divided into subsets according to the pattern of local reference), vertically (the set of attributes is partitioned according to different applications at different locations), or into arbitrary fragments, with or without overlap among them [Mahmoud et al. 1979].

Gavish and Segev [1983] describe an algorithm for optimizing set operations in horizontally partitioned relational databases. The problem is proven to be computationally untractable, and heuristics for its solution are developed. Ullman [1982] describes the use of "guard conditions" for simplifying queries on horizontally distributed databases by identifying the relevant fragments. Work on general nondisjoint fragments has just begun [Maier and Ullman 1983].

The main focus of the existing literature is on vertically distributed databases. In this case, restriction and projection can be performed locally; research has therefore concentrated on the optimal implementation and scheduling of joins and other multirelation operations. The main tool used in this context is the semijohn operation described in Section 4.3. As discussed, tree queries can be completely solved by using semijoins. For the special case of chain queries, an efficient algorithm exists that computes the optimal semijoin schedule when the reduction factors of each semijoin are known [Chiu et al. 1981]. Similar algorithms for general tree queries are given by Chiu and Ho [1980] and Gouda and Dayal [1981]. However, owing to the computational complexity of exact methods, heuristic procedures are normally preferred [Bernstein et al. 1981; Chang 1982; Cheung 1982a; Yu and Chang 1983].

Some early algorithms do not use semijoins. Wong [1977] employs a hill-climbing procedure to improve incrementally on an initial solution that processes the complete query at one site. Epstein et al. [1978] present a model for minimizing processing time or network traffic separately for each operation in a query. No global optimality is guaranteed. The optimization algorithm for R* (a distributed extension of System R [Williams et al. 1982]) performs an exhaustive search over combinations of several alternative join strategies in multijoin queries. Within its search space and the limits of data estimation, an optimal solution is found [Daniels 1982; Daniels et al. 1982; Ng 1982; Selinger and Adiba 1970].

As in the centralized case, many distributed query evaluation algorithms have been criticized for either assuming too much statistical information to be practical or for making unrealistic simplifications. Muthuswamy and Kerschberg [1983] describe a procedure for obtaining detailed database statistics by observing database usage.

The computational complexity of the distributed query optimization problem has

lead to the same phenomenon that we observed for centralized queries: One group of researchers is looking for optimal solutions in special cases, while another employs general heuristics in order to be able to answer all queries. If one plans to implement a distributed DBMS, the challenge is to combine both approaches in a homogeneous way. A good example of such a comprehensive strategy, which includes the efficient handling of fragments, data replication, and dynamic access plans (avoiding some of the aforementioned pitfalls of complicated statistical estimates), is given by Yu and Chang [1983].

6.3 Database Machines

The use of database machines is motivated primarily by the goal of relieving a general-purpose host computer from the burden of database processing so that overall system performance may be improved. Off-loading DBMS functionality from the host to a back-end database machine can be achieved with a software-oriented or hardware-oriented approach.

In the *software backend approach* (surveyed, e.g., by Maryanski [1980]), the database—together with DBMS software—is transferred to a standby general-purpose computer. The software back end completely processes each database request, thus enabling the host to execute nondatabase tasks in parallel. Another attractive property of the approach is the reasonable cost of conventional computers. For very database-intensive applications, however, the software back-end approach does not seem to be appropriate. In these situations, the back-end computer itself becomes the system bottleneck since it has the same limitations as a general-purpose host. Overall performance may be even worse than in a host-only configuration because of the additional interprocessor communication requirements.

In such a case, a *hardware backend approach* (surveyed, e.g., by Hsiao [1979] and Langdon [1979]) could be chosen. From the viewpoint of query optimization, hardware back ends can be used to support important optimization strategies, such as early evaluation of restrictive operations and parallel processing, by placing on-board logic as close as possible to the base data and dividing labor among existing hardware components.

In general, a hardware back end consists of a set of cooperating special-purpose processors. A wide bandwidth of architectural alternatives exists, ranging from the association of identical processors with disjoint fragments of the database to the assignment of specific processors to different DBMS functions; these assignments can be made either statically or dynamically.

The so-called "cellular logic" [Su 1979] is characterized by the fragmentation of the database (e.g., residing on a disk) into equal-sized cells (e.g., disk tracks), each of which is associated with a special-purpose processor. Usually, these processors have an identical repertoire and are connected with a master processor controlling the concurrent operations of its slaves. Various cellular logic prototypes, such as CASSM [Su and Lipkovsky 1975], RAP [Ozkarahan 1982], and RARES [Lin et al. 1976], have received wide attention.

Some designs of database machines have also experimented with associative arrays [Berra and Oliver 1979]. However, owing to their relatively high cost and limited capacity, they do not constitute a realistic solution for the permanent storage of entire databases. Associative arrays therefore are usually proposed to be used as staging devices for relatively inexpensive mass storage devices.

The specialization of processors to specific DBMS functions has been realized in a number of database machine architectures. Specific cost models [Bernadat 1983] and optimization algorithms [Valduriez and Gardarin 1984] have been developed. In the DBC [Banerjee and Hsiao 1979], access control, directory maintenance, and query processing are performed by different computers. The dynamic assignment of single processors to query-processing tasks, such as the evaluation of distinct subqueries, has been pursued in DIRECT [DeWitt 1979], as well as the search processor of the Technical University of Braunschweig [Leilich et al. 1978]. More recently, LSI

and VLSI technology have been explored for the purposes of close integration of data storage and query-processing capabilities, as well as hardware implementations of language constructs usually available in high-level query languages [Kim et al. 1981; Menon and Hsiao 1981].

7. SUMMARY

An overview of logical transformation techniques and physical evaluation methods for database queries 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 processing queries efficiently in conventional centralized and distributed database systems.

Query optimization research is still an active field. Promising directions include the development of simple yet realistic cost estimates, the optimization of queries on databases with deductive or computational capabilities, and the simultaneous optimization of multiple queries and update transactions. Other interesting areas only briefly addressed in this survey are query optimization in database systems that utilize more advanced access paths, such as multiple-attribute indexes or database machines, and query optimization in systems that work on the complex data structures required for artificial intelligence, office, statistical, decision support, or computer-aided design and manufacture applications.

ACKNOWLEDGMENTS

The work of Matthias Jarke was supported in part by a joint study with the IBM Corporation. The work of Jürgen Koch was supported in part by the Deutsche Forschungsgemeinschaft under Grant SCHM450/2-1 (principal investigator: J. W. Schmidt). The authors are grateful to Joachim Schmidt and Arnie Rosenthal for many discussions and suggestions, and to the referees and editors for helpful comments on the presentation of this material.

REFERENCES

AHO, A. V., BEERI, C., AND ULLMAN, J. D. 1979a. The theory of joins in relational databases. *ACM Trans. Database Syst. 4,* 3 (Sept.), 297–314.

AHO, A. V., SAGIV, Y., AND ULLMAN, J. D. 1979b. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst. 4,* 4 (Dec.), 435–454.

AHO, A. V., SAGIV, Y., AND ULLMAN, J. D. 1979c. Equivalences among relational expressions. *SIAM J. Comput. 8,* 2, 218–246.

ANDLER, S., DING, I., ESWARAN, K., HAUSER, C., KIM, W., MEHL, J., AND WILLIAMS, R. 1982. System D: A distributed system for availability. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 33–44.

APERS, P. M. G., HEVNER, A. R., AND YAO, S. B. 1983. Optimization algorithms for distributed queries. *IEEE Trans. Softw. Eng. SE-9,* 1, 57–68.

ASTRAHAN, M. M., AND CHAMBERLIN, D. D. 1975. Implementation of a structured English query language. *Commun. ACM 18,* 10 (Oct.), 580–588.

ASTRAHAN, M. M., AND GHOSH, S. P. 1974. A search path selection algorithm for the Data Independent Accessing Model (DIAM). In *Proceedings of the ACM-SIGMOD Workshop on Data Description, Access and Control* (Ann Arbor, Mich., May 1–3). ACM, New York, pp. 367–388.

ASTRAHAN, M. M., SCHKOLNICK, M., AND KIM, W. 1980. Performance of the System R access path selection algorithm. In *Information Processing 80.* Elsevier North-Holland, New York, pp. 487–491.

BANCILHON, F., RICHARD, P., AND SCHOLL, M. 1982. On line processing of compacted relations. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 263–269.

BANERJEE, J., AND HSIAO, D. K. 1979. DBC—A database computer for very large databases. *IEEE Trans. Comput. C-28,* 6, 414–429.

BATORY, D. S. 1982. Index selection. In *Principles of Database Design,* S. B. Yao, Ed. Springer, New York.

BAYER, R., AND MCCREIGHT, E. 1972. Organization and maintenance of large ordered indexes. *Acta Inf. 1,* 173–189.

BAYER, R., ELHARDT, K., KIESSLING, W., AND KILLAR, D. 1984. Verteilte Datenbanksysteme. *Inf. Spektrum 7,* 1, 1–19.

BENTLEY, J. L., AND FRIEDMAN, J. H. 1979. Data structures for range searching. *ACM Comput. Surv. 11,* 4 (Dec.), 397–409.

BERNADAT, P. 1983. Decomposition et évaluation de questions dans une machine base de données relationnelles. Thèse IIIme cycle, Départment d'Informatique, Université de Paris VI, Paris, France.

BERNSTEIN, P. A., AND CHIU, D. M. W. 1981. Using semi-joins to solve relational queries. *J. ACM 28,* 1 (Jan.), 25–40.

BERNSTEIN, P. A., AND GOODMAN, N. 1981a. The power of natural semijoins. *SIAM J. Comput. 10,* 4, 751–771.

BERNSTEIN, P. A., AND GOODMAN, N. 1981b. The power of inequality semijoins. *Inf. Syst. 6,* 4, 255–265.

BERNSTEIN, P. A., AND GOODMAN, N. 1981c. Concurrency control in distributed database systems. *ACM Comput. Surv. 13,* 2 (June), 185–221.

BERNSTEIN, P. A., BLAUSTEIN, B. T., AND CLARKE, E. M. 1980. Fast maintenance of semantic integrity assertions using redundant aggregate data. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 126–136.

BERNSTEIN, P. A., GOODMAN, N., WONG, E., REEVE, C. L., AND ROTHNIE, J. B., JR. 1981. Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Syst. 6,* 4 (Dec.), 602–625.

BERRA, B., AND OLIVER, E. 1979. The role of associative array processors in database machine architectures. *IEEE Comput. 12,* 3, 53–61.

BITTON, D., BORAL, H., DEWITT, D., AND WILKINSON, W. K. 1983. Parallel algorithms for the execution of relational database operations. *ACM Trans. Database Syst. 8,* 3 (Sept.), 324–353.

BLASGEN, M. W., AND ESWARAN, K. P. 1976. On the evaluation of queries in a relational data base system. IBM Res. Rep. RJ 1745, IBM Research Laboratories, San Jose, Calif.

BLASGEN, M. W., AND ESWARAN, K. P. 1977. Storage and access in relational databases. *IBM Syst. J. 16,* 363–377.

BOLOUR, A. 1981. Optimal retrieval for small range queries. *SIAM J. Comput. 10,* 4, 721–741.

BREITBART, Y., AND REITER, A. 1975. Algorithms for fast evaluation of Boolean expressions. *Acta Inf. 4,* 107–116.

BRODIE, M., MYLOPOULOS, J., AND SCHMIDT, J. W., Eds. 1984. *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases, and Programming Languages.* Springer, New York.

BUNEMAN, P. 1979. The problem of multiple paths in a database schema. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 368–372.

CARDENAS, A. F. 1975. Analysis and performance of inverted data base structures. *Commun. ACM 18,* 5 (May), 253–263.

CARLIS, J. V., MARCH, S. T., AND DICKSON, G. W. 1981. Physical database design: A DSS approach. In *Proceedings of the 2nd International Conference on Information Systems* (Boston, Mass.). ACM, New York, pp. 153–172.

CARLSON, C. R., AND KAPLAN, R. S. 1976. A generalized access path model and its application to a relational data base system. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Washington, D.C., June 2–4). ACM, New York, pp. 143–154.

CERI, S., AND PELAGATTI, G. 1982. Allocation of operations in distributed database access. *IEEE Trans. Comput. C-31,* 2, 119–128.

CHAMBERLIN, D. D., ASTRAHAN, M. M., KING, W. F., LORIE, R. A., MEHL, J. W., PRICE, T. G., SCHKOLNICK, M., SELINGER, P. G., SLUTZ, D. R., WADE, B. W., AND YOST, R. A. 1981. Support for repetitive transactions and ad hoc queries in System R. *ACM Trans. Database Syst. 6,* 1 (Mar.), 70–94.

CHAN, A., AND NIAMIR, B. 1982. On estimating cost of accessing records in blocked database organizations. *Comput. J. 25,* 3, 368–374.

CHAN, A., DAYAL, U., FOX, S., GOODMAN, N., RIES, D. D., AND SKEEN, D. 1983. Overview of an Ada compatible distributed database manager. In *SIGMOD 83, Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 228–237.

CHANDRA, A. K., AND HAREL, D. 1982a. Structure and complexity of relational queries. *J. Comput. Syst. Sci. 25,* 99–128.

CHANDRA, A. K., AND HAREL, D. 1982b. Horn clauses and the fixpoint query hierarchy. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31). ACM, New York, pp. 158–163.

CHANDRA, A. K., AND MERLIN, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computation* (Boulder, Colo., May 24). ACM, New York, pp. 77–90.

CHANG, C. L. 1978. DEDUCE 2: Further investigations of deduction in relational databases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 210–236.

CHANG, C. L. 1979. On evaluation of queries containing derived relations in a relational data base. IBM Res. Rep. RJ2667, IBM Research Laboratories, San Jose, Calif.

CHANG, J.-M. 1982. A heuristic approach to distributed query processing. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 54–61.

CHEN, P. P. S., AND AKOKA, J. 1980. Optimal design of distributed information systems. *IEEE Trans. Comput. C-29,* 12, 1068–1080.

CHESNAIS, A., GELENBE, E., AND MITRANI, I. 1983. On the modeling of parallel access to shared data. *Commun. ACM 26,* 3 (Mar.), 196–202.

CHEUNG, T.-Y. 1982a. A method for equijoin queries in distributed relational databases. *IEEE Trans. Comput. C-31,* 8, 746–751.

CHEUNG, T.-Y. 1982b. Estimating block accesses and number of records in file management. *Commun. ACM 25,* 7 (July), 484–487.

CHIU, D. M., AND HO, Y. C. 1980. A methodology for interpreting tree queries into optimal semi-join expressions. In *Proceedings of the ACM—SIGMOD International Conference on Management of Data* (Santa Monica, Calif., May 14–16). ACM, New York, pp. 169–178.

CHIU, D. M., BERNSTEIN, P. A., AND HO, Y. C. 1981. Optimizing chain queries in a distributed database system. Tech. Rep. TR-01-81, Computer

Science Dept., Harvard University, Cambridge, Mass.

CHRISTODOULAKIS, S. 1981. Estimating selectivities in data bases. Tech. Rep. CSRG-136, Computer Science Dept., University of Toronto, Toronto, Canada.

CHRISTODOULAKIS, S. 1983. Estimating block transfers and join sizes. In *SIGMOD 83, Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York. pp. 40–54.

CHU, W. W., AND HURLEY, P. 1982. Optimal query processing for distributed database systems. *IEEE Trans. Comput. C-31,* 9, 835–850.

CLAUSEN, S. E. 1980. Optimizing the evaluation of calculus expressions in a relational database system. *Inf. Syst. 5,* 1, 41–54.

CODD, E. F. 1971. A database sublanguage founded on the relational calculus. In *Proceedings of the ACM–SIGFIDET Workshop, Data Description, Access, and Control* (San Diego, Calif., Nov. 11–12). ACM, New York, pp. 35–68.

CODD, E. F. 1972. Relational completeness of data base sublanguages. In *Courant Computer Science Symposia No. 6: Data Base Systems.* Prentice-Hall, New York, pp. 67–101.

DANIELS, D. 1982. Query compilation in a distributed database system. IBM Res. Rep. RJ3423, IBM Research Laboratories, San Jose, Calif.

DANIELS, D., SELINGER, S., HAAS, L., LINDSAY, B., MOHAN, C., WALKER, A., AND WILMS, P. 1982. An introduction to distributed query compilation in R*. In *Proceedings of the 2nd Symposium on Distributed Databases* (Berlin, FRG). Elsevier North-Holland, New York.

DAVIS, H. W., AND WINSLOW, L. E. 1982. Computational power in query languages. *SIAM J. Comput. 11,* 3, 547–554.

DAVIS, L. S., AND KUNII, T. L. 1982. Pattern databases. In *Data Base Design Techniques II,* S. B. Yao and T. L. Kunii, Eds. Springer, New York, pp. 357–399.

DAYAL, U. 1983a. Evaluating queries with quantifiers: A horticultural approach. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Atlanta, Ga.). ACM, New York, pp. 125–136.

DAYAL, U. 1983b. Processing queries over generalization hierarchies in a multi-database system. In *Proceedings of the 9th International Conference on Very Large Data Bases* (Florence, Italy). VLDB Endowment, Saratoga, Calif., pp. 342–353.

DAYAL, U., AND GOODMAN, N. 1982. Query optimization for CODASYL database systems. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 138–150.

DAYAL, U., GOODMAN, N., LANDERS, T. A., OLSON, K., SMITH, J. M., AND YEDWAB, L. 1981. Local query optimization in Multibase—A system for heterogeneous distributed databases. Tech. Rep. CCA-81-11, Computer Corporation of America, Cambridge, Mass.

DEMOLOMBE, R. 1980. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *Proceedings of the 6th Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 55–63.

DEWITT, D. J. 1979. Query execution in DIRECT. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 13–22.

DOWNEY, P. J., SETHI, R., AND TARJAN, R. E. 1980. Variations on the common subexpression problem. *J. ACM 27,* 4 (Oct.), 758–771.

EGGERS, S. J., AND SHOSHANI, A. 1980. Efficient access of compressed data. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 205–211.

EPSTEIN, R., AND STONEBRAKER, M. 1980. Analysis of distributed data base processing strategies. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 92–101.

EPSTEIN, R., STONEBRAKER, M., AND WONG, E. 1978. Distributed query processing in a relational data base system. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Austin, Tex., May 31–June 2). ACM, New York, pp. 169–180.

ESCULIER, C., AND CLORIEUX, A. M. 1979. The SIRIUS-DELTA distributed DBMS. In *Proceedings of the International Conference on Entity-Relationship Approach,* P. Chen, Ed. Elsevier North-Holland, New York, pp. 543–551.

ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. 1976. The notions of consistency and predicate locks in a database system. *Commun. ACM 19,* 11 (Nov.), 624–633.

FINKELSTEIN, S. 1982. Common expression analysis in database applications. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 235–245.

FORKER, H. J. 1982. Algebraical and operational methods for the optimization of query processing in distributed relational database management systems. In *Proceedings of the 2nd International Symposium on Distributed Databases* (Berlin, FRG). Elsevier North-Holland, New York, pp. 39–59.

GARDARIN, G., VALDURIEZ, P., AND VIEMONT, Y. 1984. Predicate trees: An approach to optimize relational query operations. In *Proceedings of the IEEE COMPDEC Conference* (Los Angeles, Calif.). IEEE, New York.

GAVISH, B., AND SEGEV, A. 1982. Query optimization in distributed computer systems. In *Management of Distributed Data Processing,* J. Akoka, Ed. Elsevier North-Holland, New York, pp. 233–252.

GAVISH, B., AND SEGEV, A. 1983. Set query optimization in horizontally partitioned distributed databases. Working Paper QM8304, Graduate School of Management, University of Rochester, Rochester, N.Y.

GELENBE, E., AND GARDY, D. 1982. The size of projections of relations satisfying a functional dependency. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 325–333.

GILLES, J. H., AND SCHUSTER, S. A. 1975. Query execution and index selection for relational data bases. Tech. Rep. CSRG-53, Computer Science Dept., University of Toronto, Toronto, Ontario.

GOODMAN, N., AND SHMUELI, O. 1980. Nonreducible database states for cyclic queries. Tech. Rep. TR-15-80, Computer Science Dept., Harvard University, Cambridge, Mass.

GOODMAN, N., AND SHMUELI, O. 1982. Tree queries: A simple class of relational queries. *ACM Trans. Database Syst. 7,* 4 (Dec.), 653–677.

GOTLIEB, L. R. 1975. Computing joins of relations. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (San Jose, Calif., May 14–16). ACM, New York, pp. 55–63.

GOUDA, M. G., AND DAYAL, U. 1981. Optimal semijoin schedules for query processing in local distributed database systems. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1). ACM, New York, pp. 164–175.

GRANT, J., AND MINKER, J. 1981. Optimization in deductive and conventional relational database systems. In *Advances in Database Theory,* H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 195–234.

GRAY, P. M. D. 1981. Use of automatic programming and simulation to facilitate operations on CODASYL databases. In *Database,* M. P. Atkinson, Ed. Pergamon Infotech, London, pp. 315–369.

GRAY, P. M. D. 1984. Implementing the join operation on CODASYL DBMS. In *Databases: Role and Structure,* P. M. Stocker, Ed. Cambridge University Press, Cambridge, England.

GRIES, D. 1971. *Compiler Construction for Digital Computers.* Wiley, New York.

GRIFFETH, N. D. 1978. Nonprocedural query processing for databases with access paths. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Austin, Tex., May 31–June 2). ACM, New York, pp. 160–168.

GRIFFITHS, P. G., AND WADE, B. W. 1976. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1,* 3 (Sept.), 242–255.

GRISHMAN, R. 1978. The simplification of retrieval requests generated by question-answering systems. In *Proceedings of the 4th International Conference on Very Large Data Bases* (West Berlin, FRG, Sept. 13–15). IEEE, New York, pp. 400–406.

GUDES, E., AND REITER, A. 1973. On evaluating Boolean expressions. *Softw. Pract. Exper. 3,* 345–350.

HALL, P. A. V. 1974. Common subexpression identification in general algebraic systems. Tech. Rep. UKSC 0060, IBM UK Scientific Center, Peterlee, England.

HALL, P. A. V. 1976. Optimization of single expressions in a relational data base system. *IBM J. Res. Devel. 20,* 3, 244–257.

HAMMER, M., AND NIAMIR, B. 1979. A heuristic approach to attribute partitioning. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 93–101.

HAMMER, M., AND ZDONIK, S. B., JR. 1980. Knowledge-based query processing. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 137–147.

HANANI, M. Z. 1977. An optimal evaluation of Boolean expressions in an online query system. *Commun. ACM 20,* 5 (May), 344–347.

HAWTHORN, P. 1982. Microprocessor assisted tuple access, decompression, and assembly for statistical database systems. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 223–233.

HENSCHEN, L. J., AND NAQVI, S. A. 1984. On compiling queries in recursive first-order databases. *J. ACM 31,* 1 (Jan.), 47–85.

HEVNER, A. R. 1979. The optimization of query processing on distributed database systems. Ph.D. dissertation, Computer Science Dept., Purdue University, West Lafayette, Ind.

HEVNER, A. R., AND YAO, S. B. 1979. Query processing on a distributed database. *IEEE Trans. Softw. Eng. SE-5,* 3, 177–187.

HEVNER, A. R., AND YAO, S. B. 1981. Transaction optimization on a distributed database system. Tech. Rep. HR-81-257, Honeywell Corporate Computer Center, Bloomington, Minn.

HSIAO, D. K. 1979. Database machines are coming, database machines are coming. *IEEE Comput. 12,* 3, 7–9.

IBM CORPORATION 1966. Introduction to IBM direct-access storage devices and organization methods. Programming manual GC 20-1649-06,

IEEE 1982. Special issue on query optimization. *IEEE Database Eng. 5,* 3 (Sept.).

JARKE, M. 1984. Common subexpression isolation in multiple query optimization. In *Query Processing in Database Systems,* W. Kim, D. Reiner, and D. Batory, Eds. Springer, New York.

JARKE, M., AND KOCH, J. 1983. Range nesting: A fast method to evaluate quantified queries. In *SIGMOD 83, Proceedings of Annual Meeting* (San

Jose, Calif., May 23–26). ACM, New York, pp. 196–206.

JARKE, M., AND SCHMIDT, J. W. 1981. Evaluation of first-order relational expressions. Tech. Rep. 78, Fachbereich Informatik, Universitaet Hamburg, Hamburg, FRG.

JARKE, M., AND SCHMIDT, J. W. 1982. Query processing strategies in the PASCAL/R relational database management system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 256–264.

JARKE, M., AND VASSILIOU, Y. 1984. Coupling expert systems with database management systems. In *Artificial Intelligence Applications for Business*, W. Reitman, Ed. Ablex, Norwood, N.J., pp. 65–85.

JARKE, M., CLIFFORD, J., AND VASSILIOU, Y. 1984. An optimizing Prolog front-end to a relational query system. In *SIGMOD 84, Proceedings of Annual Meeting* (Boston, Mass., June 18–21). ACM, New York, pp. 296–306.

JOHNSON, D. S., AND KLUG, A. 1982. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31). ACM, New York, pp. 164–169.

JOHNSON, D. S., AND KLUG, A. 1983. Optimizing conjunctive queries that contain untyped variables. *SIAM J. Comput. 12*, 4, 616–640.

JOHNSTON, H. R., SCHWEITZER, J. E., AND WARKENTINE, E. R. 1983. A DBMS facility for handling structured engineering entities. In *Proceedings of the Database Week Engineering Design Applications Conference* (San Jose, Calif.). ACM, New York, pp. 3–12.

KAMBAYASHI, Y., AND YOSHIKAWA, M. 1983. Query processing utilizing dependencies and horizontal decomposition. In *SIGMOD 83 Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 55–67.

KAMBAYASHI, Y., YOSHIKAWA, M., AND YAJIMA, S. 1982. Query processing for distributed databases using generalized semi-joins. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 151–160.

KATZ, R. H., AND WONG, E. 1982. Decompiling CODASYL DML into relational queries. *ACM Trans. Database Syst. 7*, 1 (Mar.), 1–23.

KELLOGG, C. 1982. A practical amalgam of knowledge and data base technology. In *Proceedings of the National Conference on Artificial Intelligence* (Pittsburgh, Pa.). AAAI, Menlo Park, Calif.

KERSCHBERG, L., TING, P. D., AND YAO, S. B. 1982. Query optimization in star computer networks. *ACM Trans. Database Syst. 7*, 4 (Dec.), 678–711.

KIM, W. 1980. A new way to compute the product and join of relations. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Santa Monica, (Calif., May 14–16). ACM, New York, pp. 179–187.

KIM, W. 1981. Query optimization for relational database systems. IBM Res. Rep. RJ3081, IBM Research Laboratories, San Jose, Calif.

KIM, W. 1982. On optimizing an SQL-like nested query. *ACM Trans. Database Syst. 7*, 3 (Sept.), pp. 443–469.

KIM, W. 1984. Global optimization of relational queries: A first step. In *Query Processing in Database Systems*, W. Kim, D. Reiner, and D. Batory, Eds. Springer, New York.

KIM, W., KUCK, D. J., AND GARSKI, D. 1981. A bit-serial/tuple-parallel relational query processor. IBM Res. Rep. RJ3194, IBM Research Laboratories, San Jose, Calif.

KING, J. J. 1979. Exploring the use of domain knowledge for query processing efficiency. Tech. Rep. STAN-CS-79-781, Computer Science Dept., Stanford University, Stanford, Calif.

KING, J. J. 1981. QUIST: A system for semantic query optimization in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 510–517.

KLUG, A. 1980. Calculating constraints on relational expressions. *ACM Trans. Database Syst. 5*, 3 (Sept.), 260–290.

KLUG, A. 1982a. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM 29*, 3 (July), 699–717.

KLUG, A. 1982b. Access paths in the "ABE" statistical query facility. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 161–173.

KLUG, A. 1983. Locking expressions for increased database concurrency. *J. ACM 30*, 1 (Jan.), 36–54.

KOCH, J., SCHMIDT, J. W., AND WUNDERLICH, V. 1981. Type derivation for first-order relational expressions. Tech. Rep. no. 79, Fachbereich Informatik, Universität Hamburg, Hamburg, FRG.

KOWALSKI, R. 1981. Logic as a database language. Unpublished manuscript, Computer Science Dept., Imperial College, London.

KUNIFUJI, S., AND YOKOTA, H. 1982. Prolog and relational databases for Fifth Generation Computer Systems. In *Proceedings of the Workshop on Logical Bases for Data Bases* (Toulouse, France). ONERA-CERT, Toulouse, France.

LAMERSDORF, W. 1984. Recursive data models for non-conventional database applications. In *Proceedings of the IEEE COMPDEC Conference* (Los Angeles, Calif.). IEEE, New York.

LANG, T., WOOD, C., AND FERNÁNDEZ, I. B. 1977. Database buffer paging in virtual storage systems. *ACM Trans. Database Syst. 2*, 4 (Dec.), 339–351.

LANGDON, G. G. 1979. Database machines: an introduction. *IEEE Trans. Comput. C-28*, 6, 381–383.

LEILICH, H.-O., STIEGE, G., AND ZEIDLER, H. C. 1978. A search processor for data base management systems. In *Proceedings of the 4th International Conference on Very Large Data Bases* (West Berlin, FRG, Sept. 13–15). IEEE, New York, pp. 280–287.

LIN, C. S., SMITH, D. C. P., AND SMITH, J. M. 1976. The design of a rotating associative memory for relational database applications. *ACM Trans. Database Syst. 1,* 1 (Mar.), 53–65.

LIU, J. W. S. 1976. Algorithms for parsing search queries in systems with inverted file organizations. *ACM Trans. Database Syst. 1,* 4 (Dec.), 299–316.

LUK, W. S. 1983. On estimating block accesses in database organizations. *Commun. ACM 26,* 11 (Nov.), 945–947.

MAEKAWA, M. 1982. Parallel join and sorting algorithms. In *Data Base Design Techniques II,* S. B. Yao and T. L. Kunii, Eds. Springer, New York, pp. 266–296.

MAHMOUD, S. A., RIORDON, J. S., AND TOTH, K. C. 1979. Database partitioning and query processing. In *Proceedings of the IFIP Working Conference on Database Architecture.* Elsevier North-Holland, New York, pp. 3–21.

MAIER, D. 1983. *The Theory of Relational Databases.* Computer Science Press, Rockville, Md.

MAIER, D., AND ULLMAN, J. D. 1983. Fragments of relations. In *SIGMOD 83, Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 15–22.

MAIER, D., AND WARREN, D. S. 1981. Incorporating computed relations in relational databases. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1). ACM, New York, pp. 176–187.

MAKINOUCHI, A., TEZUKA, M., KITAKAMI, H., AND ADACHI, S. 1981. The optimization strategy for query evaluation in RDB/V1. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 518–529.

MALL, M., REIMER, M., AND SCHMIDT, J. W. 1984. Data selection, sharing and access control in a relational scenario. In *On Conceptual Modeling. Perspectives from Artificial Intelligence, Databases, and Programming Languages,* M. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds. Springer, New York, pp. 411–436.

MARBURGER, H., AND NEBEL, B. 1983. Natuerlichsprachlicher Datenbankzugang mit HAM-ANS: Syntaktische Korespondenz, natuerlichsprachliche Quantifizierung und semantisches Modell des Diskursbereichs. In *Sprachen fuer Datenbanken,* J. W. Schmidt, Ed. Springer-Verlag, Berlin, pp. 26–41.

MARCH, S. T. 1983. A mathematical programming approach to the selection of access paths for large multiuser data bases. *Decision Sci. 14,* 4, 564–587.

MARYANSKI, F. J. 1980. Backend database systems. *ACM Comput. Surv. 12,* 1 (Mar.), 3–25.

MENON, M. J., AND HSIAO, D. K. 1981. Design and analysis of a relational join operation for VLSI. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 44–55.

MERRETT, T. H. 1977. Database cost analysis: A top-down approach. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Toronto, Canada, Aug. 3–5). ACM, New York, pp. 135–143.

MERRETT, T. H. 1981. Why sort-merge gives the best implementation of the natural join. *SIGMOD Rec. 13,* 2, 39–51.

MERRETT, T. H., KAMBAYASHI, Y., AND YASUURA, H. 1981. Scheduling of page-fetches in join operations. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 488–498.

MINKER, J. 1975. Performing inferences over relation data bases. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (San Jose, Calif., May 14–16). ACM, New York, pp. 79–91.

MINKER, J. 1978. Search strategy and selection function for an inferential relational system. *ACM Trans. Database Syst. 3,* 1 (Mar.), 1–31.

MINKER, J., AND NICOLAS, J.-M. 1983. On recursive axioms in deductive databases. *Inf. Syst. 8,* 1, 1–13.

MISSIKOFF, M., AND SCHOLL, M. 1983. Relational queries in domain based DBMS. In *SIGMOD 83, Proceedings of Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 219–227.

MONTGOMERY, A. I., D'SOUZA, D. J., AND LEE, S. B. 1983. The cost of relational algebraic operations in skewed data: Estimates and experiments. In *Information Processing 83.* Elsevier North-Holland, New York, pp. 235–241.

MUNZ, R. R. 1979. Gross architecture of the distributed database system. VDN. In *Proceedings of the IFIP Working Conference on Database Architecture.* Elsevier North-Holland, New York, pp. 23–34.

MUNZ, R. R., SCHNEIDER, H.-J., AND STEYER, F. 1979. Application of sub-predicate tests in database systems. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 426–435.

MUTHUSWAMY, B., AND KERSCHBERG, L. 1983. Distributed query optimization using detailed database statistics. Unpublished manuscript, Computer Science Dept., University of South Carolina.

NAU, D. 1983. Expert computer systems. *IEEE Comput. 16,* 2 (Feb.), 63–85.

NEUHOLD, E. J., AND BILLER, H. 1977. POREL: A distributed data base on an inhomogeneous computer network. In *Proceedings of the 3rd Inter-*

national Conference on Very Large Data Bases (Tokyo, Oct. 6–8). IEEE, New York, pp. 380–395.

NG, P. 1982. Distributed compilation and recompilation of distributed queries. IBM Res. Rep. RJ3375, IBM Research Laboratories, San Jose, Calif.

NIEBUHR, K. E., AND SMITH, S. E. 1976. *N*-ary joins for processing Query by Example. *IBM Tech. Disclosure Bull. 19,* 6, 2377–2381.

NIEBUHR, K. E., SCHOLZ, K. W., AND SMITH, S. E. 1976. Algorithm for processing Query by Example. *IBM Tech. Disclosure Bull. 19,* 2, 736–741.

NIEVERGELT, J., HINTERBERGER, H., AND SEVCIK, K. C. 1984. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst. 9,* 1 (Mar.), 38–71.

NILSSON, N. 1982. *Principles of Artificial Intelligence.* Springer, New York.

OTT, N. 1977. Interpretation of questions with quantifiers and negation in the USL system. IBM Tech. Rep. TR-77.10.005, IBM Scientific Center, Heidelberg, FRG.

OTT, N., AND HORLAENDER, K. 1982. Removing redundant join operations in queries involving views. IBM Tech. Rep. TR-82.03.003, IBM Scientific Center, Heidelberg, FRG.

OZKARAHAN, E. A. 1982. Database machine/computer based distributed databases. In *Proceedings of the 2nd International Symposium on Distributed Databases* (Berlin, FRG). Elsevier North-Holland, New York, pp. 61–80.

OZSOYOGLU, M. Z., AND OZSOYOGLU, G. 1983. An extension of relational algebra for summary tables. In *Proceedings of the 2nd Statistical Database Workshop* (Berkeley, Calif.). University of California, Berkeley.

OZSOYOGLU, M., AND YU, C. T. 1980. On identifying a class of database queries that can be processed efficiently. In *Proceedings of the IEEE COMPSAC Conference.* IEEE, New York, pp. 453–461.

PAIGE, R. 1982. Applications of finite differencing to database integrity control and query/transaction optimization. In *Proceedings of the Workshop on Logical Bases for Data Bases* (Toulouse, France). ONERA-CERT, Toulouse, France.

PALERMO, F. P. 1972. A data base search problem. In *Proceedings of the 4th Symposium on Computer and Information Science* (Miami Beach, Fla.). AFIPS Press, Reston, Va., pp. 67–101.

PARSAYE, K. 1983. Database management, knowledge base management, and expert system development in PROLOG. In *Proceedings of the Database Week Conference on Engineering Applications of Databases* (San Jose, Calif.). ACM, New York, pp. 159–178.

PECHERER, R. M. 1975. Efficient evaluation of expressions in a relational algebra. In *Proceedings of the ACM Pacific 75 Conference* (San Francisco, Calif., May 14–16). ACM, New York, pp. 44–49.

PECHERER, R. M. 1976. Efficient exploration of product spaces. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Washington, D.C., June 2–4). ACM, New York, pp. 169–177.

PIROTTE, A. 1979. Findamental and secondary issues in the design of non-procedural relational languages. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 239–250.

PUTKONEN, A. 1979. On the selection of the access path in inverted database organizations. *Inf. Syst. 4,* 4, 219–225.

REIMER, M. 1983. Solving the phantom problem by predicative optimistic concurrency control. In *Proceedings of the 9th International Conference on Very Large Data Bases* (Florence, Italy). VLDB Endowment, Saratoga, Calif., pp. 81–88.

REITER, R. 1978. Deductive question-answering on relational data bases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 149–178.

RICHARD, P. 1981. Evaluation of the size of a query expressed in relational algebra. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1). ACM, New York, pp. 155–163.

ROSENKRANTZ, D. J., AND HUNT, H. B. III. 1980. Processing conjunctive predicates and queries. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 64–72.

ROSENTHAL, A., AND REINER, D. 1982. An architecture for query optimization. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 246–255.

ROSENTHAL, A., AND REINER, D. 1984. Querying relational views of networks. In *Query Processing in Database Systems,* W. Kim, D. Reiner, and D. Batory, Eds. Springer, New York.

ROTHNIE, J. B. 1974. An approach to implementing a relational data management system. In *Proceedings of the ACM–SIGMOD Workshop on Data Description, Access, and Control* (Ann Arbor, Mich., May 1–3). ACM, New York, pp. 277–294.

ROTHNIE, J. B., JR. 1975. Evaluating inter-entry retrieval expressions in a relational data base management system. In *Proceedings of the National Computer Conference* (Anaheim, Calif., May 19–22), vol. 44. AFIPS Press, Reston, Va., pp. 417–423.

ROTHNIE, J. B., AND GOODMAN, N. 1977. A survey of research and development in distributed database management. In *Proceedings of the 3rd International Conference on Very Large Data Bases* (Tokyo, Oct. 6–8). IEEE, New York, pp. 48–62.

ROUSSOPOULOS, N. 1982a. View indexing in relational databases. *ACM Trans. Database Syst. 7,* 2 (June), 258–290.

ROUSSOPOULOS, N. 1982b. The logical access path schema of a database. *IEEE Trans. Softw. Eng. SE-8,* 6, 563–573.

SACCO, G. M., AND SCHKOLNICK, M. 1982. A mechanism for managing the buffer pool in a relational database system using the hot set model. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 257–262.

SACCO, G. M., AND YAO, S. B. 1982. Query optimization in distributed database systems. In *Advances in Computers,* vol. 21. Academic Press, New York, pp. 225–273.

SAGALOWICZ, D. 1977. IDA: An intelligent data access program. In *Proceedings of the 3rd International Conference on Very Large Data Bases* (Tokyo, Oct. 6–8). IEEE, New York, pp. 293–302.

SAGIV, Y. 1981. *Optimization of Queries in Relational Databases.* UMI Research Press, Ann Arbor, Michigan.

SAGIV, Y. 1983. Quadratic algorithms for minimizing joins in restricted relational expressions. *SIAM J. Comput. 12,* 2, 316–328.

SAGIV, Y., AND YANNAKAKIS, M. 1980. Equivalences among relational expressions with the union and difference operators. *J. ACM 27,* 4 (Oct.) 633–655.

SALTON, G., AND WONG, A. 1978. Generation and search of clustered files. *ACM Trans. Database Syst. 3,* 4 (Dec.) 321–346.

SCHEK, H.-J., AND PISTOR, P. 1982. Data structures for an integrated database management and information retrieval system. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 197–207.

SCHENK, K. L., AND PINKERT, J. R. 1977. An algorithm for servicing multi-relational queries. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Toronto, Canada, Aug. 3–5). ACM, New York, pp. 10–20.

SCHKOLNICK, M. 1975. The optimal selection of indexes for files. *Inf. Syst. 1,* 4, 141–146.

SCHKOLNICK, M. 1982. Physical database design techniques. In *Data Base Design Techniques II,* S. B. Yao and T. L. Kunii, Eds. Springer, New York, pp. 229–252.

SCHMIDT, J. W. 1977. Some high level language constructs for data of type relation. *ACM Trans. Database Syst. 2,* 3 (Sept.), 247–261.

SCHMIDT, J. W. 1979. Parallel processing of relations: A single-assignment approach. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 398–408.

SCHMIDT, J. W. 1984. Database programming: Language constructs and execution models. In *Programmiersprachen und Programmentwicklung,* U. Ammann, Ed. Springer-Verlag, Berlin, pp. 1–26.

SELINGER, P. G., AND ADIBA, M. 1980. Access path selection in distributed database systems. IBM Res. Rep. RJ2283, IBM Research Laboratories, San Jose, Calif.

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. 1979. Access path selection in a relational database management system. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 23–34.

SEVERANCE, D. G., AND CARLIS, J. V. 1977. A practical approach to selecting record access paths. *ACM Comput. Surv. 9,* 4 (Dec.) 259–272.

SHMUELI, O. 1981. The fundamental role of tree schemas in relational query processing. Ph.D. thesis, Computer Science Dept., Harvard Univ., Cambridge, Mass.

SHNEIDERMAN, B. 1977. Reduced combined indexes for efficient multiple attribute retrieval. *Inf. Syst. 2,* 4, 149–154.

SHNEIDERMAN, B., AND GOODMAN, V. 1976. Batched searching of sequential and tree structured files. *ACM Trans. Database Syst. 1,* 3 (Sept.), 268–275.

SHOSHANI, A. 1982. Statistical database: Characteristics, problems, and some solutions. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 208–222.

SHULTZ, R. K., AND ZINGG, R. J. 1984. Response time analysis of multiprocessor computers for database support. *ACM Trans. Database Syst. 9,* 1 (Mar.), 100–132.

SMITH, J. M., AND CHANG, P. Y. T. 1975. Optimizing the performance of a relational algebra database interface. *Commun. ACM 18,* 10 (Oct.), 568–579.

SMITH, J. M., BERNSTEIN, P. A., DAYAL, U., GOODMAN, N., LANDERS, T., LIN, K. W. T., AND WONG, E. 1981. MULTIBASE—Integrating heterogeneous distributed database systems. In *Proceedings of the AFIPS National Computer Conference* (Chicago, May 4–7), vol. 50. AFIPS Press, Reston, Va., pp. 487–499.

SOCKUT, G. H., AND GOLDBERG, R. P. 1979. Database reorganization—Principles and practice. *ACM Comput. Surv. 11,* 4 (Dec.) 371–395.

STONEBRAKER, M. 1975. Implementation of integrity constraints and views by query modification. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (San Jose, Calif., May 14–16). ACM, New York, pp. 65–78.

STONEBRAKER, M., AND NEUHOLD, E. 1977. A distributed database version of INGRES. In *Proceedings of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.). University of California, Berkeley.

STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. 1976. The design and implementation of INGRES. *ACM Trans. Database Syst. 1,* 3 (Sept.), 189–222.

STROET, J. W. M., AND ENGMANN, R. 1979. Manipulation of expressions in a relational algebra. *Inf. Syst. 4,* 4, 195–203.

SU, S. Y. W. 1979. Cellular-logic devices: Concepts and applications. *IEEE Comput. 12,* 3, 11–25.

SU, S. Y. W., AND LIPKOVSKI, G. 1975. CASSM: A cellular system for very large databases. In *Proceedings of the 1st International Conference on Very Large Data Bases* (Framingham, Mass., Sept. 22–24). ACM, New York, pp. 456–472.

SU, S. Y. W., AND MIKKILINENI, K. P. 1982. Parallel algorithms and their implementation in MICRONET. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 310–324.

TANENBAUM, A. 1981. *Computer Networks.* Prentice-Hall, New York.

TEOREY, T. J., AND FRY, J. P. 1982. *Design of Database Structures.* Prentice-Hall, New York.

TODD, S. 1974. Implementing the join operator in relational data bases. IBM Scientific Center Tech. Note 15, IBM UK Scientific Center, Peterlee, England.

TSICHRITZIS, D. 1976. LSL: A link and selector language. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Washington, D.C., June 2–4). ACM, New York, pp. 123–133.

ULLMAN, J. D. 1982. *Principles of Database Systems.* Computer Science Press, Rockville, Md.

VALDURIEZ, P. 1982. Semi-join algorithms for multiprocessor systems. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 225–233.

VALDURIEZ, P., AND GARDARIN, G. 1984. Join and semijoin algorithms for a multiprocessor database machine. *ACM Trans. Database Syst. 9,* 1 (Mar.), 133–161.

VAN DE RIET, R. P., WASSERMAN, A. I., KERSTEN, M. L., AND DE JONGE, W. 1981. High-level programming features for improving the efficiency of a relational database system. *ACM Trans. Database Syst. 6,* 3 (Sept.), 464–485.

VASSILIOU, Y., AND JARKE, M. 1984. Query languages—A taxonomy. In *Human Factors and Interactive Computer Systems,* Y. Vassiliou, Ed. Ablex, Norwood, N.J.

VASSILIOU, Y., AND LOCHOVSKY, F. 1980. DBMS transaction translation. In *Proceedings of the IEEE COMPSAC Conference.* IEEE, New York, pp. 89–96.

VASSILIOU, Y., CLIFFORD, J., AND JARKE, M. 1984. Access to specific declarative knowledge in expert systems: The impact of logic programming. *Decision Support Syst. 1,* 1.

VERHOFSTAD, J. S. M. 1978. Recovery techniques for database systems. *ACM Comput. Surv. 10,* 2 (June), 167–195.

WALKER, A. 1980. On retrieval from a small version of a large data base. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 47–54.

WARREN, D. H. D. 1981. Efficient processing of interactive relational database queries expressed in logic. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 272–281.

WELCH, J. W., AND GRAHAM, J. W. 1976. Retrieval using ordered lists in inverted and multilist files. In *Proceedings of the ACM–SIGMOD International Conference on Management of Data* (Washington, D.C., June 2–4). ACM, New York, pp. 21–29.

WHANG, K.-Y., WIEDERHOLD, G., AND SAGALOWICZ, D. 1983. Estimating block accesses in database organizations: A closed noniterative formula. *Commun. ACM 26,* 11 (Nov.), 940–944.

WILLIAMS, R., DANIELS, D., HAAS, L., LAPIS, G., LINDSAY, B., NG, P., OBERMARCK, R., SELINGER, P., WALKER, A., WILMS, P., AND YOST, R. 1982. R*: An overview of the architecture. In *Proceedings of the International Conference on Database Systems* (Jerusalem, Israel).

WONG, E. 1977. Retrieving dispersed data from SDD-1: A system for distributed databases. In *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.), pp. 217–235.

WONG, E. 1983. Dynamic rematerialization: Processing distributed queries using redundant data. *IEEE Trans. Softw. Eng. SE-9,* 3, 228–232.

WONG, E., AND KATZ, R. H. 1983. Distributing a database for parallelism. In *SIGMOD 83, Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 23–29.

WONG, E., AND YOUSSEFI, K. 1976. Decomposition—A strategy for query processing. *ACM Trans. Database Syst. 1,* 3 (Sept.), 223–241.

XU, G. D. 1983. Search control in semantic query optimization. Tech. Rep. #83-09, Computer and Information Science Dept., University of Massachusetts, Amherst, Mass.

YANG, C.-S. 1977. Avoiding redundant accesses in unsorted multilist file organizations. *Inf. Syst. 2,* 4, 155–158.

YAO, S. B. 1977a. Approximating block accesses in database organizations. *Commun. ACM 20,* 4 (Apr.), 260–261.

YAO, S. B. 1977b. An attribute based model for database access cost analysis. *ACM Trans. Database Syst. 2,* 1 (Mar.), 45-67.

YAO, S. B. 1979. Optimization of query evaluation algorithms. *ACM Trans. Database Syst. 4,* 2 (June), 133–155.

YAO, S. B., AND DEJONG, D. 1978. Evaluation of database access paths. In *Proceedings of the ACM–SIGMOD International Conference on*

Management of Data (Austin, Tex., May 31–June 2). ACM, New York, pp. 66–77.

YOUSSEFI, K., AND WONG, E. 1979. Query processing in a relational database management system. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 409–417.

YU, C. T., AND CHANG, C. C. 1983. On the design of a query processing strategy in a distributed database environment. In *SIGMOD 83, Proceedings of the Annual Meeting* (San Jose, Calif., May 23–26). ACM, New York, pp. 30–39.

YU, C. T., AND OZSOYOGLU, M. 1979. An algorithm for tree query membership of a distributed query. In *Proceedings of the IEEE COMPSAC Conference*. IEEE, New York, pp. 306–312.

YU, C. T., LUK, W. S., AND SIU, M. K. 1978. On the estimation of the number of desired records with respect to a given query. *ACM Trans. Database Syst. 3*, 1 (Mar.), 41–56.

ZANIOLO, C. 1979. Design of relational views over network schemas. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 179–190.

Received October 1983; final revision accepted April 1984.