INTRODUCTION TO QUERY PROCESSING

Matthias Jarke
Graduate School of Business Administration
New York University

Jürgen Koch, Joachim W. Schmidt
Fachbereich Informatik, Johann Wolfgang Goethe-Universitat
Dantestr. 9, 6000 Frankfurt 1, West Germany

March 1984

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

Working Paper Series

CRIS #73

GBA #84-48(CR)

Published in W. Kim, D. Reiner, D. Batory (eds.), Query Processing in Database Systems, Springer-Verlag, 1984.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

INTRODUCTION TO QUERY PROCESSING

Abstract. Query processing in databases can be divided into two steps: selecting an 'optimal' evaluation strategy, and executing it. We first present elementary nested loop and relational algebra algorithms for query execution and point out some opportunities for improving their performance. A survey of optimization strategies, structured in query transformation techniques and access planning methods, follows. Finally, extensions for special-purpose query systems are briefly addressed.

1.0 PERFORMANCE CONSIDERATIONS IN DATABASE SYSTEMS

Database management systems (DBMS) are now a widely accepted tool for reducing the problem of managing a large collection of shared data for application programmers and end users. The user interacts with the system by submitting requests for data selection (queries) or manipulation (updates). Both kinds of operations frequently involve access to data described to the system in terms of their properties rather than their location. A sequence of queries or updates which is logically a single unit of interaction with the database is called a transaction.

To fulfill its mission, a DBMS must be efficient in the sense that it minimizes the consumption of human and machine resources for processing transactions submitted to it. The costs of human resources in utilizing a DBMS are determined, among other factors, by the power and friendliness of the language provided to each type of user (application programmer or end user), and by the system's response time. The goals of language power and fast response time may be in conflict since it is often difficult to implement a powerful language construct efficiently. It is the task of the database implementor to reduce this potential problem.

Machine resources used by the DBMS include the storage space for data and access paths in secondary memory, as well as for main memory buffers, and the time spent by the CPU and channels for data transfer to and from secondary memory and other computers (in distributed databases). The trade-off between these cost components is influenced by the architecture of the database system.

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

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

In a geographically distributed DBMS with relatively slow communication lines between the sites where data reside and the sites where requests originate, communication delay dominates the costs while the other factors are only relevant for local suboptimization. In centralized systems, the emphasis is on minimizing secondary storage accesses (transfer channel usage), although for complex queries the CPU costs may also be quite high. Finally, in locally distributed DBMS's, all factors have similar weights resulting in very complex cost functions.

There is also a higher-level trade-off between user and machine cost components [APER83]. An effort to minimize response time is reasonable only under the assumption that user time is the most important bottleneck resource. Otherwise, direct cost minimization of machine resource usage can be attempted. Fortunately, user and machine-oriented goals are largely complementary; where goal conflicts arise, they can often be resolved by assigning limits to the availability of machine resources (e.g., main memory buffer space).

Exact optimization of these cost factors is usually not only computationally infeasible but also prevented by the lack of precise database statistics, i.e., information about the size of data objects and the distribution of data values. Nevertheless, it is customary to use the term <u>query optimization</u> for the heuristic selection of strategies to improve the efficiency of executing individual queries. Database management systems can support the achievement of efficiency by providing the following subsystems:

1. a physical design environment which allows the physical structure of the database to be adapted to an expected usage pattern [MARC84];

2. a transaction management mechanism that allows multiple access sequences to be executed concurrently but without mutual interference that would lead to inconsistent data [GRAY81];

3. a query processing subsystem that evaluates queries efficiently within the constraints created by the two previous mechanisms.

This chapter addresses the question of how to construct a query processor for a relational DBMS (other types of database systems will be considered briefly). We first discuss how high-level language queries can be represented in the DBMS (section 2). Next, we contrast two elementary algorithms for processing a given query, and present examples and a general framework for improving their efficiency (section 3). In sections 4 and 5, two basic strategies within this framework will be investigated: the transformation of a query into a form that can be evaluated more efficiently, and the generation of a good access plan for the fast evaluation of a given representation form. Environments where conventional query processing is not sufficient will be reviewed in section 6.

2.0 QUERIES AND QUERY LANGUAGES

Many user interfaces can be constructed on top of the same database system. This paper will use a <u>relational framework</u>. We briefly review relational data structures and integrity constraints before focusing our attention on the representation of relational queries; for more background on the relational model of data, the reader is referred to the literature (e.g., [MAIE83], [ULLM82]). In the relational model, <u>data</u> are organized in tables or <u>relations</u>. The columns of the tables are called <u>attributes</u>; all values appearing in an attribute are elements of a common <u>domain</u>. The rows of the tables are called records, <u>tuples</u>, or simply relation elements.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

In addition to these structural properties, relational databases must often satisfy certain semantic integrity constraints. For example, a frequent type of integrity constraint has the format: "if any two tuples of relation R agree in attributes A1, ..., Am, then they must also agree in attributes B1, ..., Bn." In this case, we say that A1, ... Am functionally determine B1, ..., Bn. Moreover, if B1, ..., Bn represents all attributes of R, we say that A1, ..., Am form a key of R, provided there is no proper subset of A1, ..., Am that functionally determines B1, ..., Bn.

A relational database schema and examples of a query formulated in a number of popular query languages are provided in Figure 2-1. The database (which will be used throughout this paper) describes EMPLOYEEs, the DEPARTMENTs and managers they work for, and the OFFICEs they are using. One employee can have several offices and each office can be occupied by several employees; the OFFICE-USE relation describes the assignment of employees to offices.

Relational Database Schema (keys are underlined):

```
EMPLOYEE   (eno, ename, marstat, salary, dno)
DEPARTMENT(dno, dname, mgr)
OFFICE     (floor, room, capacity)
OFFICE-USE(eno, floor, room)
```

Example Query in English:

names of single employees in the computer department who make less than $40000.-

SQL:

```
SELECT ename
FROM   EMPLOYEE
WHERE  salary < 40000 AND marstat = single AND
       dno = (SELECT dno
              FROM DEPARTMENT
              WHERE dname = 'computer')
```

QUEL:

```
RANGE OF e IS EMPLOYEE
RANGE OF d IS DEPARTMENT
RETRIEVE (e.ename) WHERE
          e.salary < 40000 AND e.marstat = single AND
          e.dno = d.dno AND d.dname = 'computer'
```

Query by Example:

EMPLOYEE	eno	ename	marstat	salary	dno
		p.	single	<40000	15

DEPARTMENT	dno	dname	mgr
	15	computer	

Figure 2-1: Examples of end user query languages

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

Query interfaces like the ones shown in Figure 2-1 may cater to different groups of database users (novices or experts, casual or frequent). For query processing purposes, it is useful to map all of these interfaces into a common intermediate language and have the query processor deal only with that language. Such a language should be powerful enough to express a large class of queries. It should also have a well-defined theoretical basis in order to allow the query processor to specify efficiency-oriented query transformations. If very powerful end user interfaces must be supported, it may be necessary to provide full programming capabilities with the intermediate language -- a database programming language [SCHM83]. This paper will describe query processing methods in the framework of the (tuple) relational calculus, integrated into the database programming language, Pascal/R [SCHM77]. This language is not meant to be a user-friendly query language for end users but allows for a uniform description of most existing query processing methods.

The <u>relational calculus</u> [CODD72] is a non-procedural notation for defining a query result through the description of its properties. The representation of a query in relational calculus consists of two parts: target list and selection expression. The <u>selection expression</u> 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 universal quantifiers). The <u>target list</u> defines the free variables occurring in the predicate, and specifies the structure of the resulting relation. The reader can use the following example to relate the relational calculus representation to his or her favorite query language from Figure 2-1.

<u>Example 2-1:</u>

Names of single employees in the computer department
who make less than $40000.

```
[<e.ename> OF EACH e IN EMPLOYEE:
                    e.salary < 40000 AND e.marstat = single
                    AND
                    SOME d IN DEPARTMENT
                        (d.dno = e.dno AND d.dname = 'computer')]
```

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

The selection expression following the colon defines constraints on the free variable. First, two <u>restrictive terms</u> determine that only single employees with a salary of under $40000 are of interest. Second, a quantified subexpression must be satisfied (read: "there exists some DEPARTMENT tuple, say d, such that..."). A <u>join term</u>, relating EMPLOYEEs to their DEPARTMENTs, is AND-connected to another restrictive term that restricts attention to computer departments. The comparison operators usually allowed in terms are =, <>, <, >, <=, and >=.

The relational calculus allows variables to be bound to different range relations. For example, variable e is bound to EMPLOYEE and variable d is bound to DEPARTMENT. In addition to the logical operator AND, the operators OR and NOT can also be used in predicates. The following two examples illustrate more complex queries, using universal quantifiers and multiple tuple variables over one relation.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

Example 2-2:

Names of departments where all employees earn less than $40000.

```
[<d.dname> OF EACH d IN DEPARTMENT:
              ALL e IN EMPLOYEE
                 (e.salary < 40000 OR e.dno <> d.dno)]
```

Example 2-3:

Employees who make less than $40000 and have an office on the same floor where their manager has one.

```
[EACH e IN EMPLOYEE: e.salary < 40000 AND
 SOME empoff IN OFFICE-USE
     (empoff.eno = e.eno
      AND
      SOME d IN DEPARTMENT
          (d.dno = e.dno
           AND
           SOME mgroff IN OFFICE-USE
               (mgroff.floor = empoff.floor AND mgroff.eno = d.mgr)))]
```

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

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

where M is a quantifier-free predicate (i.e., a Boolean expression) called the matrix. For instance, queries expressed in QUEL (see Figure 2-1) are always in prenex normal form. If, furthermore, M is of the form

(T11 AND ... AND T1k) OR ... OR (Tm1 AND ... AND Tmk)

(where the Tij are terms) the query is said to be in disjunctive prenex normal form (DPNF). The query in Example 2.2 is in DPNF while those in the other two examples are not. The set of all Tij for a given i is called the i-th conjunction of the matrix; a query which contains only one conjunction is called a conjunctive query [CHAN77].

In [CODD72] the relational calculus was 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. Relational completeness has to be considered a minimum requirement. An often-cited example for a conceptually simple query which goes beyond relational completeness is "find the employees reporting to manager Smith at any level." Furthermore, users often request aggregated summary data which cannot be described in pure relational calculus. For example, a query for "offices with free capacity" requires a count function over the relation OFFICE-USE to be computed. However, the extension of relational calculus by aggregate functions is rather straightforward [KLUG82a].

Thus far, we have considered queries in their role as requests by end users. Queries are also used as part of update transactions which change the stored data based on their current value. For example, an update request, "raise by 5% the salaries in all departments where nobody earns more than $40000.-", would involve answering the query given in Example 2-2. Moreover, query-like expressions are used internally in a DBMS to express integrity constraints or access rights [STON75]. Such a constraint might be: "a manager is entitled to at least one non-shared office." An 'intelligent' DBMS could apply this constraint to rephrase a query for "offices with free capacity" in a way that does not count space in the private offices of managers.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

3.0 QUERY PROCESSING AND A GENERAL OPTIMIZATION FRAMEWORK

There have been two principal approaches to constructing a general query evaluation algorithm for relational databases: the direct interpretation of calculus expressions as nested loop procedures, and the translation of typical subexpression patterns into operations of a relational algebra. In this section, we review both approaches and then state a general framework, in which improvements to each procedure and hybrids between them can be described. The direction of such improvements is indicated by means of examples.

3.1 Nested Loop Solutions

Any query processing algorithm must state how the target list and the selection expression of a query will be evaluated. The most straightforward algorithm translates the relational calculus query into a nested loop. For describing this procedure, we employ a PASCAL-like database programming language which offers a FOR EACH construct that retrieves single tuples in system-determined sequence, and can evaluate quantifier-free Boolean expressions. The language also provides mechanisms to declare relational variables, to assign values (relations) to them, and to insert new subrelations using the operator, :+.

A query of the form

```
[<f1, ..., fn> OF EACH r IN rel:  pred(r)]
```

translates to the program:

```
result : RELATION OF RECORD f1: ...; fn: ... END;
BEGIN
 result := [];  (* the empty relation *)
 FOR EACH r IN rel DO
  IF bool(pred(r)) THEN result :+ [<r.f1, ..., r.fn>]
END
```

This extends easily to the case of more than one variable in the target list. The quantifier-free Boolean expression, bool(pred(r)), is derived recursively from the quantified selection predicate, pred(r), by creating Boolean functions for each quantifier in pred(r), as indicated in the following example.

<u>Example 3-1:</u>

The query of Example 2-1 would translate into the program:

```
result : RELATION OF RECORD ename: ... END;

FUNCTION some-d(e) : Boolean;
BEGIN
 some-d := false;
 FOR EACH d IN DEPARTMENT DO
  some-d := some-d OR d.dname = 'computer' AND d.dno = e.eno
END;

BEGIN
 result := [];
 FOR EACH e IN EMPLOYEE DO
  IF e.salary < 40000 AND e.marstat = single AND some-d(e)
  THEN result :+ [<e.ename>]
END.
```

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

A closer look at this simple procedure reveals a number of efficiency problems which should be solved by query optimization methods. Four points of attack will be mentioned, some of which will be studied in more detail later.

1. The semantics of quantifiers can be taken into account when implementing the functions. For example, the loop in function some-d could stop, after the first DEPARTMENT tuple satisfying both conditions has been retrieved:

```
FUNCTION some-d(e) : Boolean;
BEGIN
 reset(DEPARTMENT);
 REPEAT read(DEPARTMENT)
 UNTIL eor(DEPARTMENT) OR
       DEPARTMENT^.dname = 'computer' AND
       DEPARTMENT^.dno = e.dno;
 some-d := NOT(eor(DEPARTMENT))
END;
```

2. If indexes or other fast access paths are available, the implementation of the function, some-d, can make use of them. For example, if a primary index exists for the DEPARTMENT relation, only one access to the corresponding DEPARTMENT tuple is required for each EMPLOYEE tuple.

3. The method does not fully utilize available buffer space. Modern computer systems retrieve data from secondary storage in blocks rather than tuple-by-tuple, and can often keep more than one block in main memory simultaneously. This can be exploited by executing the algorithm block-wise rather than record-at-a-time [KIM80], possibly in conjunction with buffer management strategies [SACC82].

4. Each call of the function, some-d, retrieves all tuples of the DEPARTMENT relation (until one qualifies, at least). It can be seen from the expression that only DEPARTMENT tuples with dname='computer' can possibly qualify. It may therefore be useful to first extract the corresponding subrelation, and then have the function, some-d, work on that subrelation rather than on the complete DEPARTMENT relation.

Generalizations of these ideas can be found in many query optimization algorithms. For example, the <u>decomposition</u> algorithm used in INGRES [WONG76] combines a general nested loop procedure (called 'tuple substitution') with the pre-evaluation of separable subexpressions as in the last strategy mentioned (called 'detachment').

3.2 Algebraic Solutions

Translating a query into a sequence of high-level operations provides a widely used alternative to nested loop algorithms. The <u>relational algebra</u> [CODD72] includes general set operations as well as specialized relational operators. The <u>restriction</u> operator evaluates a query whose selection expression contains one restrictive term. For example,

```
RESTRICT (DEPARTMENT, dname='computer') =
        [EACH d IN DEPARTMENT: d.dname = 'computer']
```

The <u>projection</u> operator constructs a vertical subset of a relation:

```
PROJECT (EMPLOYEE, [ename]) =
        [<e.ename> OF EACH e IN EMPLOYEE: true]
```

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

The join operator permits two relations with at least one comparable attribute to be combined into one, e.g.,

```
JOIN (EMPLOYEE, dno = dno, DEPARTMENT) =
     [EACH e IN EMPLOYEE, EACH d IN DEPARTMENT: e.eno = d.dno]
```

If no restriction is placed on the combination of tuples, the join degenerates to a Cartesian product.

Example 3-2:

The complete query of Example 2-1 corresponds to

```
PROJECT(RESTRICT(RESTRICT(JOIN(EMPLOYEE,
                               dno = dno,
                               RESTRICT(DEPARTMENT, dname = 'computer')),
                          salary < 40000),
                 marstat = single),
        [ename])
```

Note that the existential quantification of the variable, d, is evaluated by applying a projection operator to the result of the join. Similarly, a more complex operation called division can be used for universal quantification.

We give below a general translation algorithm introduced in [CODD72] and refined by [PALE72]. It translates a relational calculus query given in DPNF to a sequence of algebra operations. The query from Example 2-1 serves as an illustration. Note the production and manipulation of major intermediate results that distinguishes algebraic methods from pure nested loop solutions.

1. Evaluate restrictive and join terms applying restriction and join operations to the range relations of the variables involved.

```
int1 := RESTRICT (DEPARTMENT, dname='computer')
int2 := JOIN (EMPLOYEE, dno=dno, DEPARTMENT)
int3 := RESTRICT (EMPLOYEE, salary<40000)
int4 := RESTRICT (EMPLOYEE, marstat=single)
```

2. Combine the results of step 1 for all terms appearing in one conjunction by means of join or Cartesian product operations. This step evaluates the AND-connection of terms within each conjunction.

```
int5 := JOIN (int1, dno=dno, int2)
int6 := JOIN (int5, eno=eno, int3)
int7 := JOIN (int6, eno=eno, int4)
```

3. Construct the union of the conjunction results computed in step 2. If a particular variable is missing in a certain conjunction, it can be added by another Cartesian product operation between the conjunction's result and the range relation of the missing variable. This step evaluates the OR-connection between conjunctions and thus completes evaluation of the matrix (and is therefore not required in our example).

4. Evaluate the quantifiers from right to left using projection for existentially quantified variables and division for universally quantified variables.

```
int8 := PROJECT (int7, [attributes of EMPLOYEE relation])
```

5. Evaluate the target list.

```
result := PROJECT (int8, [ename])
```

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

The algebraic approach partitions the query optimization problem into two tasks: translating the query into a 'good' sequence of operations, and optimizing the implementation of each operation. Strategies for the former subproblem will be considered in section 4. Here, we briefly address the implementation of algebra operations. In particular, by introducing the join operation we gain the freedom for considering an alternative to the nested loop solution and its derivatives: the merge join. In this method, the two relations to be joined are sorted by the same attribute and then scanned concurrently to find all pairs of matching tuples.

The implementation of merge join is slightly more complex than it would seem from this simple description. If neither of the two join attributes is a key to its relation (i.e., the join implements a many-to-many relationship), intermediate relations may have to be built. From the program sketch provided in Figure 3-1, it is evident that the choice of which is the 'inner' and the 'outer' relation will influence the size of these intermediate results. However, in Example 3-2, where dno is a key to DEPARTMENT, no intermediate relations are needed if DEPARTMENT is chosen as the 'inner' relation.

```
(* outer, inner : the two relations to be joined
   outer^, inner^ : buffers for the last read elements
   outer^.f, inner^.g : the join attributes
   current : a variable indicating the current join value
   joinresult : a relation whose attribute set is the union
                of the attribute sets of outer and inner *)
---------------------------------------------------------------
BEGIN
 sort(outer by f); sort(inner by g);
 reset(outer); reset(inner);
 read(outer); read(inner);
 joinresult := [];
 REPEAT
  WHILE NOT (eor(inner) OR eor(outer) OR outer^.f<>inner^.g) DO
   IF outer^.f < inner^.g THEN read(outer) ELSE read(inner);
  IF NOT (eor(inner) OR eor(outer))
  THEN BEGIN (* Cartesian product of joining subrelations *)
        intermediate := []; current := outer^.f;
        WHILE inner^.g = current AND NOT (eor(inner)) DO
        BEGIN
         intermediate :+ [inner^];
         read(inner)
        END;
        WHILE outer^.f = current AND NOT (eor(outer)) DO
        BEGIN
         FOR EACH irec IN intermediate DO
             joinresult :+ [<outer^, irec>];
         read(outer)
        END
       END
 UNTIL eor(outer) OR eor(inner)
END.
```

Figure 3-1: A merge (equi-)join algorithm for m:n relationships

Methods can be devised to compress the intermediate results required in algebraic methods. Attributes not appearing in the query can be removed by an initial projection operation, or a tuple identifier can substitute for a complete relation element [PALE72]. Where Cartesian product operations are required, it is even possible to represent all elements of a relation by a special value [JARK82]. However, the advantages of data compression must be traded off against the costs of decompressing the final query output.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

Naive use of the relational algebra has one severe disadvantage: it separates operations which could easily be executed in a parallel or pipelined fashion. For instance, the last two projections in the translation example above could be combined into one projection (as shown in Example 3-2). One way out of this dilemma is the explicit introduction of parallel processing [YAO79]. Alternatively, one can provide more powerful operations. Examples include the semijoin operation (see section 4.4, below), and the graft and prune operations for evaluating quantified queries proposed in [DAYA83].

3.3 Integrated Solutions: A General Optimization Framework

Many query optimization heuristics have emerged from each of the two basic query processing strategies presented in the previous subsections. Such heuristics were often developed as efficiency-enhancing add-ons to implemented DBMS. The two approaches overlap only partially in their coverage of query optimization opportunities. In addition, researchers identified classes of queries for which fast special-purpose algorithms exist. It is the task of a query optimization subsystem to identify and compare the applicable strategies for each query. However, the amount of optimization is restricted by the goal to minimize the overall costs, including the cost of the optimization itself.

There seems to be a need for an integrated framework in which all of the ideas can be brought into play in a structured manner. We utilize such a framework to organize our survey of query optimization techniques:

1. Apply logical transformations to the query representation that standardize, simplify, and ameliorate the query to streamline the evaluation and to detect applicable special-case procedures.

2. Map the transformed query into alternative sequences of operations, i.e., generate a set of candidate 'access plans'.

3. Compute the overall cost for each access plan, select the cheapest one, and execute it.

Transformation strategies are to a large degree independent of the database state at a given time, and thus can be applied mostly at compile time. The richness of the access plans generated and the optimality of the choice, however, are dependent upon the degree of knowledge about current physical database characteristics. Most of the access plan evaluation should therefore be performed at runtime; nevertheless, due to implementation difficulties, access plans are often completely generated at compile time [SELI79]. A meta-database (e.g., an augmented data dictionary) must maintain general information about the database structure and statistical information about the database contents.

4.0 TRANSFORMATION OF QUERY REPRESENTATIONS

A query can be represented in a number of semantically equivalent relational calculus expressions. Some are better suited for efficient evaluation than others. The strategies presented in this section try to convert a given expression into a better one. They standardize and simplify a query, and assign it (where possible) to a class of queries for which fast algorithms exist. Some of the transformations presented below are <u>syntactic</u> in nature; they rely on general equivalence of language expressions whose validity is independent of any particular query or database. In contrast, <u>semantic</u> transformation strategies utilize knowledge about a particular database or application, often represented by integrity constraints.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

4.1 Specialized Query Representations

While the principle underlying all of these transformations is readily explained in the relational calculus framework, special-purpose representations have been proposed in which certain transformation algorithms are easier to describe. In particular, the so-called tableau representation [AHOS79], [SAGI81] is used in the simplification of a query, whereas object and operator query graphs are mostly applied in detecting special cases of queries.

Figure 4-1 gives a tableau representation of Example 2-1. Tableaux are a tabular notation for a subset of relational calculus queries characterized by containing only AND-connected terms and no universal quantifiers. The columns of a tableau correspond to the attributes of the underlying database. The first row of the matrix serves the same purpose as the target list of a relational calculus expression. The other rows describe the predicate.

eno	ename	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	<40000	b2			EMPLOYEE
				b2	computer	b3	DEPARTMENT

Figure 4-1: Tableau representation of Example 2-1

The symbols appearing in a tableau are distinguished variables denoted ai (corresponding to free variables), nondistinguished variables denoted bj (corresponding to existentially quantified variables), constants (corresponding to the constants in restrictive terms), blanks, and tags (indicating the range relation). A join term is indicated by having the same variable appear in different rows. Tableaux serve as a convenient notation for simplifying a large class of queries; an example will be given in section 4.3.

Figure 4-2 shows an object graph for the relational calculus expression in Example 2-1, and Figure 4-3 gives an operator graph, corresponding to the equivalent relational algebra expression of Example 3-2. Nodes in object graphs represent objects, such as (relation) variables and constants. Edges describe terms that these objects are to fulfill [PALE72], [WONG76]. Operator graphs describe an operator-controlled data flow. They represent operators as nodes and connect them by edges, indicating the direction of data flow from existing data structures to the desired result [SMIT75], [YAO79].

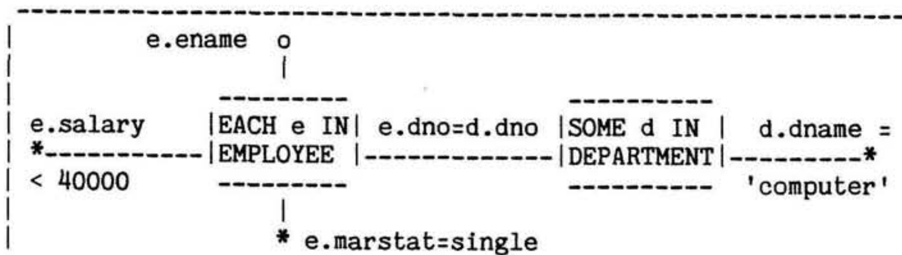

Figure 4-2: An object graph representing the calculus query.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

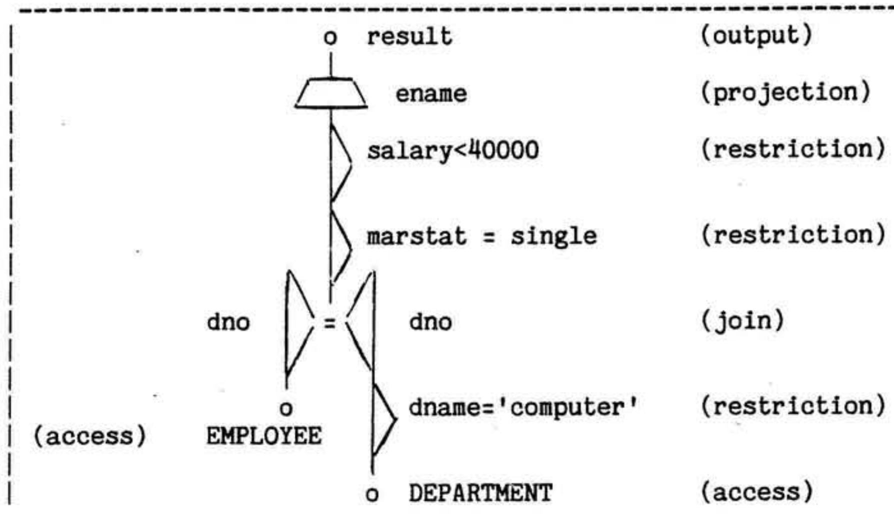

Figure 4-3: An operator graph representing the algebra query.

4.2 Standardization Of Query Representations

Most query evaluation algorithms initially transform a given query into some standard representation in order to obtain a uniform starting point from which optimization can be attempted. Most standard forms utilize prenex normal form, as introduced in section 2. For example, standardization of SQL queries [KIM82] removes the distinction between joins and nesting in that language, replacing nested expressions by joins. SDD-1 [BERN81c] and Pascal/R [JARK82] standardize relational calculus queries further into DPNF, in order to facilitate the decomposition of a query into independently evaluable subexpressions. (However, there may be cases where such a decomposition leads to unnecessary operations [GRAN81].) INGRES [WONG76] prefers a conjunctive normal form to achieve fast rejection of tuples which do not satisfy the matrix.

Example 4-1:

The standardization of the expression in Example 2-1 involves the application of a quantifier movement rule:

pred1 AND SOME r IN rel (pred2) = SOME r IN rel (pred1 AND pred2).

The repeated application of this transformation leads to the DPNF:

```
[EACH e IN EMPLOYEE:
 SOME d IN DEPARTMENT
 (e.salary < 40000 AND e.marstat = single AND
  e.dno = d.dno AND d.dname = 'computer')]
```

4.3 Simplification Of Query Representations

Even in standard form, a query may be phrased in many ways, in particular, with differing degree of redundancy. The query optimizer should try to avoid unnecessary operations caused by redundant predicates. One might argue that users are unlikely to formulate queries with redundant predicates. However, there is no assurance. Moreover, the query submitted to the DBMS may be a translation from a higher-level end user interface (e.g., a natural [OTTH82] or deductive language [REIT78]), which utilizes views defined on the schema of

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

stored database relations. Queries on views are normally translated into queries on stored relations by substituting the expression defining a view for the view identifier and renaming variables appropriately [STON75]. Such a direct translation, however, can produce unnecessarily complex queries.

Example 4-2:

Consider the following query as a direct translation from the natural language query: "what are the names of single computer people in a tax bracket under 50%?"

```
[<c.ename> OF EACH c IN compemp:
                    c.marstat = single AND SOME t IN tax50 (t.eno = c.eno)]
```

Let the views, compemp and tax50, be defined as:

```
compemp = [EACH e IN EMPLOYEE:
           SOME d IN DEPARTMENT (e.dno = d.dno AND d.dname = 'computer')]

tax50   = [EACH e IN EMPLOYEE:
                e.marstat = single  AND e.salary < 40000
                OR
                e.marstat = married AND e.salary < 80000]
```

After view substitution and standardization, the query becomes

```
[<c.ename> OF EACH c IN EMPLOYEE:
 SOME d IN DEPARTMENT SOME t IN EMPLOYEE
 (c.dno = d.dno AND d.dname = 'computer' AND c.marstat = single AND
  t.marstat = single AND t.salary < 40000 AND c.eno = t.eno
  OR
  c.dno = d.dno AND d.dname = 'computer' AND c.marstat = single AND
  t.marstat = married AND t.salary < 80000 AND c.eno = t.eno)]
```

Would the reader recognize our simple standard example in this monster? Yet, this is precisely the task of query simplification. For expressions containing no universal quantifiers, tableau techniques can be used for simplification. Such a query can be broken down into conjunctive subqueries. Thus, two tableaux result from Example 4-2 (Figure 4-4).

eno	ename	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	b2	b3			EMPLOYEE
				b3	computer	b4	DEPARTMENT
b1	b5	single	<40000	b6			EMPLOYEE

eno	ename	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	b2	b3			EMPLOYEE
				b3	computer	b4	DEPARTMENT
b1	b5	married	<80000	b6			EMPLOYEE

Figure 4-4: Tableaux for view query

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

The first tableau produces the same result as the one in Figure 4-1. To see why, we need to know that eno (as a key of the EMPLOYEE relation) functionally determines ename, marstat, salary, and dno in all rows tagged EMPLOYEE. Since b1 appears in in the eno columns of rows 1 and 3, we can substitute a2 for b5, <40000 for b2, and b3 for b6, such that the two rows become equal. Applying a syntactic simplification rule, pred AND pred <==> pred (which holds for any predicate -- see Figure 4-5(a)), one of the duplicate rows can subsequently be dropped; the resulting tableau equals the one in Figure 4-1, except for renaming of nondistinguished variables. The second tableau represents the empty relation. Using the same reasoning as before, the entries in the marstat column of rows 1 and 3 should be equal -- a contradiction with the actual tableau.

The relational calculus basis for syntactic simplification is given by idempotency rules (Figure 4-5(a)). A sophisticated application of such rules is complicated by the fact that they apply to arbitrary subexpressions which may be at a higher level than individual terms. A prerequisite for high-level simplification is therefore the recognition of <u>common subexpressions</u>, to which the rules can be applied. [HALL76] describes heuristics that detect common subexpressions using a bottom-up merge procedure in an operator graph.

For certain classes of queries, efficient algorithms exist that minimize the number of rows in a tableau (and thus the number of join operations in the underlying query) [AHOS79]. As the above example demonstrates, the application of semantic integrity constraints yields further opportunities for simplification [JARK84b], [OTTH82], [REIN84], [SAGI81], based on relational database theory (see, e.g., [MAIE83]). More sophisticated applications of semantic constraints also cover the access planning step (section 5), using AI-based heuristic deduction for what has been called <u>semantic query optimization</u> [KING81], [HAMM80]. Semantic rules (<u>guard conditions</u> [ULLM82]) can finally be used in horizontally distributed databases to locate relevant data, and thus to simplify queries in the sense that only sites are accessed where relevant data actually reside.

Additional opportunities for simplification arise if empty relations (Figure 4-5(b)) or the semantics of the comparison operators are considered. Of particular interest is transitivity [ROSE80]. Joins can be simplified to restrictions by <u>constant propagation</u>

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

or an expression can be proven <u>unsatisfiable</u> in cases such as

```
r.A > s.B AND s.B >= t.C AND t.C >= r.A.
```

```
------------------------------------------------------------
| (a) simplification: some idempotency rules                |
|                                                           |
| pred OR pred       <==> pred   pred AND pred      <==> pred  |
| pred OR NOT(pred)  <==> TRUE   pred AND NOT(pred) <==> FALSE |
| p1 OR (p1 AND p2)  <==> p1     p1 AND (p1 OR p2)  <==> p1    |
| pred OR FALSE      <==> pred   pred AND FALSE     <==> FALSE |
| pred OR TRUE       <==> TRUE   pred AND TRUE      <==> A     |
|-----------------------------------------------------------|
| (b) simplification: rules for empty relations             |
|                                                           |
| [<r.A1,..,r.An> OF EACH r IN []: pred]  <==> []            |
| SOME r IN [] (pred)                     <==> FALSE         |
| ALL  r IN [] (pred)                     <==> TRUE          |
------------------------------------------------------------
```

<u>Figure 4-5</u>: Simplification rules in relational calculus

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

4.4 Improvement Of Query Representations

For many queries, the choice of differing original formulations or simplification strategies may lead to different evaluation costs. Further transformations try to improve a query representation by detecting special cases, for which fast algorithms exist. In section 3.2, we observed that a sequence of projections from the same relation can be combined into one. The same holds for sequences of restriction operations. Such enhanced operations will tend to be profitable if either none or all of the participating attributes are indexed. If there is a mixture of indexed and nonindexed attributes, the difference in performance will be smaller.

Join operations are more complex than restriction or projection. It is therefore often useful to execute one-variable operations as early as possible [SMIT75] in order to reduce the input size of subsequent joins. There may be a conflict between this heuristic and the previous one; the optimal solution depends on file structures and join algorithms used by the query processor.

Example 4-3:

The algebra expression in Example 3-2 can be improved to

```
PROJECT (RESTRICT (JOIN (EMPLOYEE,
                         dno = dno,
                         RESTRICT(DEPARTMENT, dname = 'computer'),
                   salary < 40000 AND marstat = single),
         [ename])
```

and further to

```
PROJECT (JOIN (PROJECT (RESTRICT (EMPLOYEE,
                                  salary<40000 AND marstat=single),
                        [ename,dno]),
               dno = dno,
               PROJECT (RESTRICT (DEPARTMENT,
                                  dname='computer'),
                        [dno])),
         [ename])
```

In the relational calculus representation, a (partial) order can be imposed on the execution of subexpressions using so-called (range-)nested expressions [JARK83]. The range relation concept of the relational calculus is extended to include relation-valued expressions, rather than just relation names. The following transformation rules may be used to generate a nested expression.

```
[EACH r IN rel: p1 AND p2]      <==> [EACH r IN [EACH r' IN rel: p1]: p2]
 SOME r IN rel (p1 AND p2)      <==>  SOME r IN [EACH r' IN rel: p1] (p2)
 ALL  r IN rel (NOT(p1) OR p2) <==>  ALL  r IN [EACH r' IN rel: p1] (p2)
```

The object graph of a nested query contains the extended range expression in its nodes (Figure 4-6). If p1 contains only restrictive terms, nested expressions represent the heuristic of evaluating one-variable expressions first.

Example 4-4:

The second version of Example 4-3 corresponds to:

```
[<e.ename> OF EACH se IN [<e.ename, e.dno> OF EACH e IN EMPLOYEE:
                           e.salary<40000 AND e.marstat=single]:
              SOME cd IN [<d.dno> OF EACH d IN DEPARTMENT:
                                  d.dname='computer']
                 (cd.dno = se.eno)]
```

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

An interesting property of range-nested expressions is that they can be easily generalized beyond restrictive predicates. Let p2 contain a quantified subexpression over a certain variable, say s, the matrix of which (possibly after internal range nesting) consists of only one join term, linking s to r. In this case, the (extended) range expression of s can be evaluated independently and only the result of it must be passed on for processing the join term. For example, in the query of Example 4-4, we can create a (hopefully very small) list of dno's and then test the EMPLOYEE tuples only against this list, rather than against the complete DEPARTMENT relation.

The stepwise reduction approach represented by nested expressions was first introduced for non-quantified variables in the INGRES decomposition algorithm [WONG76]: if two subexpressions overlap in a single variable, one of them can be detached and evaluated separately. [YOUS79] presents experimental evidence for the advantages of this heuristic in terms of processing time. Subquery detachment has captured wide-spread attention especially in distributed databases since it may reduce considerably the amount of data transfer if the detached subexpression is executed at a different site from the rest of the query. In the algebra representation, a new operator, semijoin [BERN81a], was introduced to map the idea:

```
SEMIJOIN (rel1, f = g, rel2) =
        [EACH r1 IN rel1: SOME r2 IN rel2 (r1.f = r2.g)]
```

Thus, a semijoin is 'half of a join', i.e., its result corresponds to that of a join between rel1 and rel2, projected back on the attributes of rel1. The ideas of nested expressions, query detachment, and semijoin are closely related to the object graph representation of queries. As it turns out, a query can be completely resolved by a sequence of semijoins if and only if there exists an equivalent formulation whose object graph is a tree [GOOD82]. Examples 2-1 and 2-2 are such 'tree queries', whereas Example 2-3 is a 'cyclic' query (Figure 4-6). Techniques for recognizing and processing cyclic queries are treated in [KAMB84].

There are cycles which can be transformed into equivalent acyclic query graphs. Such cycles include those which (a) are introduced by transitivity [YUOZ79], [BERN81a]; (b) contain certain combinations of inequality join term edges [BERN81b], [OZSO80]; (c) are "closed" by universally quantified variables [JARK83]; (d) contain variables that can be decomposed by use of functional dependencies [KAMB83].

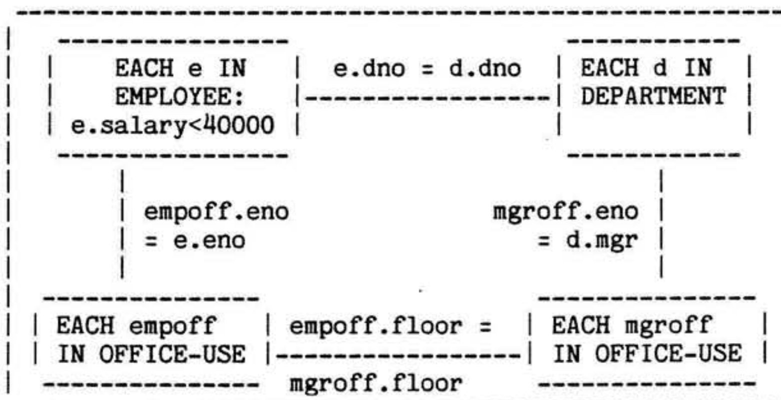

Figure 4-6: Cyclic object graph for Example 2-3

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

5.0 ACCESS PLANNING

The transformation step reduces the number of possible evaluation algorithms for a query to those for which efficient execution can be expected. Nevertheless, a large number of possibilities remain. The query optimizer can sequence the operations and it can choose the best implementation for each one. Typically, the optimization of access plans is a three-step process. The steps may be interleaved with each other and sometimes also with the transformation step.

1. Generate reasonable logical access plans, i.e., operator graphs of the improved query representation. Often, monadic operations are removed from the graph to focus on the most expensive tasks: the execution of join operations [ROSE82].

2. Augment the logical access plans by details of the physical representation of data (location of data, sort orders, existence of physical access paths, statistical information).

3. Apply a model of access and processing costs to select the cheapest access plan.

5.1 The Role Of Physical Database Structures

The evaluation of access plans depends heavily on the physical database structure. In the simplest case, the database is centralized. There may, however, still be a choice between access to base data or some auxiliary direct access structures. Additionally, the sequence of operations has to be determined in a way that minimizes the number of secondary storage accesses.

Evaluating queries over distributed databases requires the additional consideration of communication costs; the main goal becomes the reduction of data transfer between sites, even at the expense of more local processing. A typical example is the implementation of a join as a sequence of two semijoins [BERN81a] in a vertically distributed database, in which whole relations or projections thereof reside on one site. If there is also horizontal fragmentation (i.e., restrictions of a relation reside on one site [ULLM82], [GAVI82]), projection and restriction may also become distributed operations.

A general strategy for distributed query processing is the decomposition of a query into subqueries to be executed where the data reside, as contrasted to the collection of all required data at one site, where the whole query is subsequently executed [CERI82]. Subqueries can be processed in parallel on different machines; therefore, the optimizer has a choice of minimizing either response time or resource consumption (e.g., total communication delay [APER83]). If database fragments overlap [MAIE83], there is often a choice of where to retrieve certain data. The answer depends on the relative speed of processors and communication channels: do we have a homogeneous or a heterogeneous network? For example, it may make sense to have complex operations executed at a large computer even if a copy of the data is available on a local personal computer. The topology of the network may also influence the complexity of access planning. In an arbitrary network, queueing delay is a major cost factor [EPST78] which can only be influenced by global decisions.

Finally, a host computer or computer network can delegate the storage management functions of database processing to separate database machines. In the software backend approach [MARY80], the DBMS is transferred to a stand-by general-purpose computer that executes all database operations. This approach is relatively cheap and permits parallel execution of other tasks on the host

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

computer. However, in very database-intensive applications, the database machine itself, or its communication channels to the host, can become a system bottleneck.

In such a case, a hardware backend can be employed [LANG78], [OZKA82] that brings on-board logic close to the stored data in order to implement query optimization strategies, such as early evaluation of restrictions and parallelism. A hardware backend typically consists of a set of cooperating parallel processors. The division of labor between these processors can be determined by partitioning the database into cells [SU79] or by partitioning the query evaluation algorithm into functions [DEWI79]. The introduction of such hardware devices can lead to new algorithms for implementing operations, such as joins or semijoins [VALD84].

5.2 Generation And Selection Of Access Plans

Example 5-1:

Suppose our example company plans to concentrate all computer personnel on the fifth floor. Hence, managers of all computer departments with employees assigned to offices outside the fifth floor should be assembled for a meeting. Assume that each database relation resides in a different site, and that sites are connected by slow communication lines so that local processing cost can be neglected in the optimization. After applying all the transformations proposed in section 4, the access planning subsystem receives the following query:

```
[<c.mgr> OF EACH cd IN [EACH d IN DEPARTMENT:
                         d.dname = 'computer']:
          SOME e IN EMPLOYEE
             (e.dno = cd.dno AND
              SOME not5 IN [<o.eno> OF EACH o IN OFFICE-USE:
                                                o.floor <> 5]
                 (not5.eno = e.eno))]
```

Obviously, this is a simple chain query (a special case of tree queries) which can be solved from inside out. This solution requires transferring eno's of employees not working on the fifth floor, say 5000, from the OFFICE-USE site to the EMPLOYEE site, and dno's for such employees, say 100, from the EMPLOYEE site to the DEPARTMENT site. The two computer departments among these are located -- at a total cost of 5100 transfers.

Fortunately, there is an alternative strategy which also involves semijoins but does not follow the quantifier sequence directly: (a) transfer the dno's of the, say, 5 computer departments from the site of the DEPARTMENT relation to the site of the EMPLOYEE relation; (b) transfer the, say, 250 eno's and dno's of the employees working in these five departments to the OFFICE-USE site (cost is 255 if a hierarchical data representation is chosen during the transfer, 500 for a relational representation); (c) check which of these eno's relate to offices outside the fifth floor and send their (two) dno's back to the DEPARTMENT site. The total cost of this strategy is 262 (respectively 507) transfers, about 5% of the above straightforward strategy.

The example illustrates two points. First, the query optimizer must plan its access strategy and cannot rely on universal heuristics; second, to do its job, it needs information about database statistics or at least estimates. (For example, in guessing the 250 eno's above we assumed uniform distribution of employees over departments.) Under these requirements, there are several different ways to proceed.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

First, each alternative access plan can be generated and evaluated completely before executing the query. This approach can cover parallel or feedback evaluation strategies in a realistic way but the optimization effort is high. Therefore, complete planning has been proposed only for restricted environments: two-variable expressions [YAO79]; monadic operations in horizontally distributed databases [GAVI82]; completely homogeneous networks [HEVN79], [CHUH82]; chain queries [CHIU81] and tree queries [GOUD81], [CHIU80]; and star computer networks [KERS82]. Even some of these have to resort to heuristic methods for reasons of complexity. Other researchers develop heuristic methods from the beginning [CHEU82], [YUCH83]. System R and R* limit the feasible join strategies to permit exhaustive search [SELI79], [SELI80].

Practical systems often plan hierarchically, generating an access plan sequentially from subplans at various levels. For example, System R [CHAM81], [SELI79] generates a plan for a nested SQL query by optimizing each query block, and then linking all these subplans. In the distributed systems, SDD-1 [BERN81c] and Multibase [SMIT81], the levels are global access planning, and local query optimization at each site.

The cost of strategies can also be computed incrementally concurrently with their generation. This approach allows whole families of strategies with common parts to be evaluated in parallel and thus reduces the costs of optimization considerably. For example, [ROSE82] suggest retaining only the minimal-cost way to each intermediate result while discarding other ways as soon as their non-optimality is detected. A similar approach is followed by the commercial INGRES version [KOOI82].

An extension of the incremental approach is a dynamic query optimization procedure. The idea derives from the observation that, at each step in query processing, the optimizer need only select the next operation optimally. To guarantee overall optimality, only the consequences of this decision for the rest of the algorithm must be estimated. A dynamic procedure has actual information about the sizes of all its operands including intermediate results. This information can also be used to update the estimates of the remaining steps. System R re-calculates the full cost of each strategy after each operation; however, it limits the search space by allowing only one intermediate result, essentially making the operator graph look like a list structure. (As noted in section 3.3, System R also performs this procedure at compile time and, therefore, cannot make use of information about the size of intermediate results.) INGRES [YOUS79] and SDD-1 [BERN81c] permit multiple intermediate results but employ greedy heuristics with little or no lookahead beyond the next operation.

5.3 Problems Of Cost Estimation

We have demonstrated the need for quantitative information about the database. Unfortunately, such information is not very easy to obtain. In particular, the size of intermediate data structures to be retrieved or transfered is difficult to estimate. General estimation methods require detailed database statistics [MUTH83]; where they are missing, simplifying assumptions must substitute for them. A valid parameter system for estimating the size of intermediate results under such assumptions [RICH81] must have measurable input parameters but its formulas must be general enough to apply to intermediate results at any level. For estimation purposes, the database state at query time is seen as the result of a random process that generates tuples from the Cartesian product of the attribute universes (governed by integrity constraints). A universe is a finite subset of the attribute domain that covers all actual values in the database (e.g., the domain of salary (integer) is restricted to a universe by the salary range in a company). [RICH81] presents a parameter system that computes the result size for any sequence of relational

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

algebra operations, under the assumption that the size of all the universes and of each possible projection are known. Estimates for the size of projections, given the semantic constraints of the database and the sizes of universes and database relations, are derived in [GELE82].

The relationship between the size of intermediate results and the number of actual secondary storage accesses depends on the physical storage structures involved, the size of buffers, and the proportion of relation elements to be accessed. If all elements of an operand of size N have to be accessed to find the desired elements, the optimal number of secondary storage accesses is approximately N/B where B is the blocking factor of the operand. [WHAN83] estimates the number of accessed pages under random placement assumptions. If direct access is used with optimal clustering, the number of secondary storage accesses to retrieve n elements is reduced to n/B.

The traditional uniformity and randomness assumptions about value distributions and tuple placements tend to overestimate costs and thus to bias the query optimizer against the use of direct access structures [CHRI81]. However, the more sophisticated techniques require more statistical information about the database. The question of how to maintain such information with tolerable overhead is not yet fully resolved.

6.0 EXTENSIONS

Additional requirements and opportunities for query optimization arise outside the traditional framework of processing single relational queries. Some query languages permit expressions that are more powerful than those expressible in relationally complete languages [CHAN82]; others work on objects that are more complex than the flat records of the relational model. Other chapters in this book address these problems in depth; in the sequel, we provide a brief overview.

The first extension to be mentioned here is the simultaneous optimization of multiple related queries, which can follow two approaches. First, the evaluation of common subexpressions can be shared among queries; subexpressions accessing the same physical data page can do so with a single secondary storage access [SHNE76], [JARK84a]. Second, a system for multiple query optimization can invest in the creation of physical access paths, such as sorting or temporary indexes, which pay off for the batch as a whole but would not be justifiable for any single query [ROUS82]. Finally, intermediate results of some queries can be stored for later use as partial results of other queries [FINK82]. Little is known about detailed results in this area. [KIM84] and [JARK84a] describe preliminary architectures and language constructs, and a number of ongoing research projects are described in [IEEE82].

Particular needs for query optimization arise when a DBMS is interfaced with artificial intelligence systems, such as expert systems or natural language systems [REIT78]. For example, expert systems try to simulate the behavior of a human expert in a specific and usually narrow domain. With the increasing popularity of expert systems in the business world, the need arises for them to obtain information about real-world facts from a corporate database [VASS84]. Instead of duplicating the database for the expert system, coupling existing DBMS with the expert system can be attempted [JARK84c]. Once such a connection exists, it can also be employed in the reverse direction, namely to enhance an existing database system with deductive capabilities [NICO83]. Both directions of interaction, however, do not use stored data directly but require inference procedures to work on them [MINK78].

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

A reasoning task submitted to the AI system usually translates into a sequence of related database calls. Optimization techniques include: the combination of multiple tuple-oriented database calls into set-oriented operations [KUNI82], [VASS83]; the simplification of such retrieval requests [JARK84b], [OTTH82]; the reorganization of stored knowledge, with the objective of simulating improvement heuristics like the ones discussed in section 4.4 [WARR81]; and the use of intermediate results for multiple query optimization [GRAN81], [JARK84a], which is especially useful in executing recursive database calls [HENS84], [MINK83].

Several application areas require specialized data structures for efficient manipulation. CAD/CAM and text processing focus on modelling objects that are much more complex than flat records [LORI84]. One way to address this problem is to define complex structures on top of a conventional database, allowing multiple views of the structures and substructures [JOHN83]. Another approach is the extension of the traditional relational model by non-first normal form relations [SCHE82], in which attribute values can be complex data structures such as arrays or even relations, making the data model recursive [LAME84]. Query optimization for such extensions is an interesting, little researched area.

In statistical databases, data could in principle be stored as standard relations since most statistical data are represented in tables anyway. However, the size and redundancy of such tables [SHOS82], the large number of null entries for attributes that are not applicable to particular relation elements, the difficulty to distinguish between attribute names (category attributes) and attribute values, and the need for computing summary data [KLUG82b] lead to new techniques for user interfaces, data modeling, and query evaluation [EGGE80], [OZSO84], [SHOS82].

Finally, heterogeneous distributed database systems (e.g., [SMIT81]) require the submission of queries to different data models such as networks or hierarchies. The point here is to go beyond a simple translation and to address optimization issues [DAYA82], [KATZ82]. One problem in these 'navigational' models is the existence of information-bearing access paths, which may have to be used to compute a join. A complementary problem, however, is to avoid following unnecessary access paths by simplifying queries during the translation process. View processing mechanisms similar to those discussed in section 4.3 can be used for this purpose [REIN84].

REFERENCES

[AHOS79] Aho, A.V., Sagiv, Y., Ullman, J.D. "Efficient optimization of a class of relational expressions", ACM Transactions on Database Systems 4 (1979), 435-454.

[APER83] Apers, P.M.G., Hevner, A.R., Yao, S.B. "Optimization algorithms for distributed queries", IEEE Transactions on Software Engineering SE-9 (1983), 57-68.

[BERN81a] Bernstein, P.A., Chiu, D.M. "Using semi-joins to solve relational queries", Journal of the ACM 28 (1981), 25-40.

[BERN81b] Bernstein, P.A., Goodman, N. "The power of inequality semijoins", Information Systems 6 (1981), 255-265.

[BERN81c] Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L., Rothnie, J.R. "Query processing in a System for Distributed Databases (SDD-1)", ACM Transactions on Database Systems 6, (1981), 602-625.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

[CERI82] Ceri, S., Pelagatti, G. "Allocation of operations in distributed databases", IEEE Transactions on Computers C-31 (1982), 119-128.

[CHAM81] Chamberlin, D.D., Astrahan, M.M., Lorie, R.A., Mehl, J.W., Price, T.G., Schkolnick, M., Selinger, P.G., Slutz, D.R., Wade, B.W., Yost, R.A. "Support for repetitive transactions and ad-hoc queries in System R", ACM Transactions on Database Systems 6 (1981), 70-94.

[CHAN77] Chandra, A.K., Merlin, P.M. "Optimal implementation of conjunctive queries in relational databases", Proceedings 9th ACM Symposium on Theory of Computation, Boulder, Co., 77-99.

[CHAN82] Chandra, A.K., Harel, P. "Structure and complexity of relational queries", Journal of Computing System Sciences 25 (1982), 99-128.

[CHEU82] Cheung, T.-Y. "A method for equijoin queries in distributed relational databases", IEEE Transactions on Computers C-31 (1982), 746-751.

[CHIU81] Chiu, D.M., Bernstein, P.A., Ho, Y.C. "Optimizing chain queries in a distributed database system", TR-01-81, Harvard University, 1981.

[CHIU80] Chiu, D.M., Ho, Y.C. "A methodology for interpreting tree queries into optimal semi-join expressions", Proceedings ACM-SIGMOD Conference, Santa Monica 1980, 169-178.

[CHRI81] Christodoulakis, S. "Estimating selectivities in data bases", Ph.D. thesis, Univ. of Toronto, 1981.

[CHUH82] Chu, W.W., Hurley, P. "Optimal query processing for distributed database systems", IEEE Transactions on Computers C-31 (1982), 835-850.

[CODD72] Codd, E,F. "Relational completeness of data base sublanguages", in Courant Computer Science Symposia 6, Data Base Systems, Prentice Hall 1972.

[DAYA82] Dayal, U., Goodman, N. "Query optimization for CODASYL database systems", Proceedings ACM-SIGMOD Conference, Orlando 1982, 138-150.

[DAYA83] Dayal, U. "Evaluating queries with quantifiers: a horticultural approach", Proceedings ACM Symposium on Principles of Database Systems, Atlanta 1983, 125-136.

[DEWI79] DeWitt, D.J. "Query execution in DIRECT", Proceedings ACM-SIGMOD Conference, Boston 1979, 13-22.

[EGGE80] Eggers, S., Shoshani, A. "Efficient access of compressed data", Proceedings 6th VLDB Conference, Montreal 1980, 205-211.

[EPST78] Epstein, R., Stonebraker, M., Wong, E. "Distributed query processing in a relational data base system", Proceedings ACM-SIGMOD Conference, Austin 1978.

[FINK82] Finkelstein, S. "Common expression analysis in database applications", Proceedings ACM-SIGMOD Conference, Orlando, 1982, 235-245.

[GAVI82] Gavish, B., Segev, A. "Query optimization in distributed computer systems", in Akoka, J. (ed.), Management of Distributed Data Processing, North-Holland 1982, 233-252.

[GELE82] Gelenbe, E., Gardy, D. "The size of projections of relations satisfying a functional dependency", Proceedings 8th VLDB Conference, Mexico City 1982, 325-333.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

[GOOD82] Goodman, N., Shmueli, O. "Tree queries: A simple class of relational queries", ACM Transactions on Database Systems 7 (1982), 653-677.

[GOUD81] Gouda, M.G., Dayal, U.D. "Optimal semijoin schedules for query processing in local distributed database systems", Proceedings ACM-SIGMOD Conference, Ann Arbor 1981, 164-175.

[GRAN81] Grant, J., Minker, J. "Optimization in deductive and conventional relational database systems", in Gallaire, H., Minker, J., Nicholas, J.M. (eds.), Advances in Database Theory, Plenum 1981, 195-234.

[GRAY81] Gray, J. "The transaction concept: virtues and limitations", Proceedings 7th VLDB Conference, Cannes 1981, 144-154.

[HALL76] Hall, P.A.V. "Optimization of a single relational expression in a relational database", IBM Journal of Research and Develeopment 20 (1976), 244-257.

[HAMM80] Hammer, M., Zdonik, S. "Knowledge-based query processing", Proceedings 6th VLDB Conference, Montreal 1980, 137-147.

[HENS84] Henschen, L., Naqvi, S. "On compiling queries in recursive first-order databases", Journal of the ACM 31 (1984), 47-85.

[HEVN79] Hevner, A.R., Yao, S.B. "Query processing on a distributed database", IEEE Transactions on Software Engineering SE-5 (1979), 177-187.

[IEEE82] Special Issue on Query Optimization, IEEE Database Engineering 5, 3 (1982).

[JARK82] Jarke, M., Schmidt, J.W. "Query processing strategies in the PASCAL/R relational database management system", Proceedings ACM-SIGMOD Conference, Orlando 1982, 256-264.

[JARK83] Jarke, M., Koch, J. "Range nesting: a fast method to evaluate quantified queries", Proceedings ACM-SIGMOD Conference, San Jose 1983, 196-206.

[JARK84a] Jarke, M. "Common subexpression isolation in multiple query optimization", this volume.

[JARK84b] Jarke, M., Clifford, J., Vassiliou, Y. "An optimizing PROLOG front-end to a relational query system", Proceedings ACM-SIGMOD Conference, Boston 1984.

[JARK84c] Jarke, M., Vassiliou, Y. "Coupling expert systems with database management systems", in Reitman, W. (ed.), Artificial Intelligence Applications for Business, Ablex, Norwood/NJ 1984, 65-85.

[JOHN83] Johnston, H.R., Schweitzer, J.E., Warkentine, E.R. "A DBMS facility for handling structured engineering entities", Proceedings Database Week Engineering Design Applications Conference, San Jose 1983, 3-12.

[KAMB83] Kambayashi, Y., Yoshikawa, M. "Query processing utilizing dependencies and horizontal decomposition", Proceedings ACM-SIGMOD Conference, San Jose 1983, 55-67.

[KAMB84] Kambayashi, Y. "Processing cyclic queries", this volume.

[KATZ82] Katz, R., Wong, E. "Decompiling CODASYL DML into relational queries", ACM Transactions on Database Systems 7, 1 (1982), 1-23.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

[KERS82] Kerschberg, L., Ting, P.D., Yao, S.B. "Query optimization in star computer networks", ACM Transactions on Database Systems 7, 4 (1982), 678-711.

[KIM80] Kim, W. "A new way to compute the product and join of relations", Proceedings ACM-SIGMOD Conference, Santa Monica 1980, 179-187.

[KIM82] Kim, W. "On optimizing an SQL-like nested query", ACM Transactions on Database Systems 7 (1982), 443-469.

[KIM84] Kim, W. "Global optimization of relational queries: a first step", this volume.

[KING81] King, J.J. "QUIST: A system for semantic query optimization in relational data bases", Proceedings 7th VLDB Conference, Cannes 1981, 510-517.

[KLUG82a] Klug, A. "Equivalence of relational algebra and relational calculus query languages having aggregate functions", Journal of the ACM 29 (1982), 699-717.

[KLUG82b] Klug, A. "Access paths in the 'Abe' statistical query facility", Proceedings ACM-SIGMOD Conference, Orlando 1982, 161-173.

[KOOI82] Kooi, R., Frankforth, D. "Query optimization in INGRES", in [IEEE82].

[KUNI82] Kunifuji, S., Yokota, H. "Prolog and relational databases for Fifth Generation Computer Systems", Proceedings Workshop on Logical Bases for Data Bases, Toulouse, December 1982.

[LAME84] Lamersdorf, W. "Recursive data models for non-conventional database applications", Proceedings IEEE COMPDEC Conference, Los Angeles 1984.

[LANG78] Langdon, J.J. "A note on associative processors for data management", ACM Transactions on Database Systems 3 (1978), 148-158.

[LORI84] Lorie, R., Kim, W., McNabb, D., Plouffe, W., Meier, A. "Supporting complex objects in a relational system for engineering databases", this volume.

[MARC84] March, S.T. "Physical database design: techniques for improved database performance", this volume.

[MAIE83] Maier, D., The Theory of Relational Databases, Computer Science Press 1983.

[MARY80] Maryanski, F.J. "Backend database systems", ACM Computing Surveys 12 (1980), 3-26.

[MINK78] Minker, J. "Search strategy and selection function for an inferential relational system", ACM Transactions on Database Systems 3 (1978), 1-31.

[MINK83] Minker, J., Nicolas, J.-M. "On recursive axioms in deductive databases", Information Systems 8 (1983), 1-13.

[MUTH83] Muthuswamy, B., Kerschberg, L. "Distributed query optimization using detailed database statistics", unpublished manuscript, Univ. of South Carolina, May 1983.

[NICO83] Nicolas, J.-M., Yazdanian, K. "An outline of BDGEN: a deductive DBMS", in Mason, R.E. (ed.), Information Processing 83, North-Holland, Amsterdam 1983, 711-717.

[OTTH82] Ott, N., Horlaender, K. "Removing redundant joins in queries involving views", IBM Heidelberg Scientific Center Technical Report TR-82.03.003, 1982.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

[OZKA82] Ozkarahan, E.A. "Database machines/ computer-based distributed databases", Proceedings Second International Symposium on Distributed Databases, Berlin 1982, 61-80.

[OZSO80] Ozsoyoglu, M., Yu, C.T. "On identifying a class of database queries that can be processed efficiently", Proceedings IEEE COMPSAC Conference, October 1980, 453-461.

[OZSO84] Ozsoyoglu, M., Ozsoyoglu, G. "A query language for statistical databases", this volume.

[PALE72] Palermo, F.P. "A data base search problem", Proceedings 4th Computer and Information Science Symposium, Miami Beach 1972, 67-101.

[REIN84] Reiner, D., Rosenthal, A. "Querying relational views of networks", this volume.

[REIT78] Reiter, R. "Deductive question-answering on relational data bases", in Gallaire, H., Minker, J. (eds.), Logic and Databases, Plenum 1978, 149-178.

[RICH81] Richard, P. "Evaluation of the size of a query expressed in relational algebra", Proceedings ACM-SIGMOD Conference, Ann Arbor 1981, 155-163.

[ROSE80] Rosenkrantz, D.J., Hunt, M.B. "Processing conjunctive predicates and queries", Proceedings 6th VLDB Conference, Montreal 1980, 64-74.

[ROSE82] Rosenthal, A., Reiner, D. "An architecture for query optimization", Proceedings ACM-SIGMOD Conference, Orlando 1982, 246-255.

[ROUS82] Roussopoulos, N. "View indexing in relational databases", ACM Transactions on Database Systems 7 (1982), 258-290.

[SACC82] Sacco, G.M., Schkolnick, M. "A mechanism for managing the buffer pool in a relational database system using the hot set model", Proceedings 8th VLDB Conference, Mexico City 1982, 257-262.

[SAGI81] Sagiv, Y., Optimization of Queries in Relational Databases, UMI Research Press, Ann Arbor 1981.

[SCHE82] Schek, H.-J., Pistor, P. "Data structures for an integrated database management and information retrieval system", Proceedings 8th VLDB Conference, Mexico City 1982, 197-207.

[SCHM77] Schmidt, J.W. "Some high-level language constructs for data of type relation", ACM Transactions on Database Systems 2 (1977), 247-261.

[SCHM83] Schmidt, J.W., Mall, M. "Abstraction mechanisms for database programming", Proceedings ACM-SIGPLAN Symposium on Programming Language Issues in Software Systems, San Francisco 1983.

[SELI79] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, P.A., Price, T.G. "Access path selection in a relational database management system", Proceedings ACM-SIGMOD Conference, Boston 1979, 23-34.

[SELI80] Selinger, P.G., Adiba, M. "Access path selection in distributed database systems", IBM Research Report RJ2283, August 1980.

[SHNE76] Shneiderman, B., Goodman, V. "Batched searching of sequential and tree-structured files", ACM Transactions on Database Systems 1 (1976), 260-275.

[SHOS82] Shoshani, A. "Statistical database: characteristics, problems, and some solutions", Proceedings 8th VLDB Conference, Mexico City 1982, 208-222.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48

[SMIT75] Smith, J.M., Chang, P.Y.T. "Optimizing the performance of a relational algebra database interface", Communications of the ACM 18 (1975), 568-579.

[SMIT81] Smith, J.M., Bernstein, P.A., Dayal, U., Goodman, N., Landers, T., Lin, K.W.T., Wong, E. "MULTIBASE -- integrating heterogeneous distributed database systems", Proceedings AFIPS NCC 1981, 487-499.

[STON75] Stonebraker, M. "Implementation of integrity constraints and views by query modification", Proceedings ACM-SIGMOD Conference, San Jose 1975, 65-77.

[SU79] Su, S.Y. "Cellular logic devices: concepts and applications", IEEE Computer 12 (1979), 11-25.

[ULLM82] Ullman, J.D., Principles of Database Systems, Computer Science Press 1982.

[VALD84] Valduriez, P., Gardarin, G. "Join and semijoin algorithms for a multiprocessor database machine", ACM Transactions on Database Systems 9 (1984), 133-161.

[VASS83] Vassiliou, Y., Clifford, J., Jarke, M. "How does an expert system get its data?", Proceedings 9th VLDB Conference, Florence 1983, 70-72.

[VASS84] Vassiliou, Y., Clifford, J., Jarke, M. "Database access requirements of knowledge-based systems", this volume.

[WARR81] Warren, D.H.D. "Efficient processing of interactive relational data base queries expressed in logic", Proceedings 7th VLDB Conference, Cannes 1981, 272-283.

[WHAN83] Whang, K.-Y., Wiederhold, G., Sagalowicz, D. "Estimating block accesses in database organizations: a closed noniterative formula", Communications of the ACM 26 (1983), 940-944.

[WONG76] Wong, E., Youssefi, K. "Decomposition - a strategy for query processing", ACM Transactions on Database Systems 1 (1976), 223-241.

[YAO79] Yao, S.B. "Optimization of query evaluation algorithms", ACM Transactions on Database Systems 4 (1979), 133-155.

[YOUS79] Youssefi, K., Wong, E. "Query processing in a relational database management system", Proceedings 5th VLDB Conference, Rio de Janeiro 1979, 409-417.

[YUCH83] Yu, C.T., Chang, C.C. "On the design of a query processing strategy in a distributed database environment", Proceedings ACM-SIGMOD Conference, San Jose 1983, 30-39.

[YUOZ79] Yu, C.T., Ozsoyoglu, M. "An algorithm for tree query membership of a distributed query", Proceedings IEEE COMPSAC Conference, November 1979, 306-312.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-84-48