QUERY PROCESSING FOR A RELATIONAL DATABASE SYSTEM

by

Karel A. Allen Youssefi

Memorandum No. UCB/ERL M78/3

6 January 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Research sponsored by the Joint Services Electronics Program Contract F44620-76-C-0100 and the Army Research Office Grant DAAG29-76-G-0245.

Query Processing for a Relational Database System

Ph.D. Karel A. Allen Youssefi EECS

Signature [illegible]
Chairman of Committee

ABSTRACT

The problem of efficiently processing queries in a relational database management system is examined. The major areas investigated are: 1) transformations of the query statement prior to processing, 2) strategies which take advantage of the structure of the query, and 3) techniques which depend upon the data referenced by a particular query.

First, a query is examined to determine what characteristics lend themselves to more efficient processing methods. Then it is shown that transformations similar to those used in compiler optimization can be applied to the initial query prior to any data accesses to achieve certain of these characteristics at a small cost.

Since the most expensive queries tend to be those which involve several relations, several techniques for processing these multi-relation queries are examined. The first is tuple substitution which is essentially equivalent to creating the entire cross product a tuple at a time. Then the

idea of reduction is introduced. Reduction takes advantage of the structure of the query to break apart a single multi-relation query into a sequence of queries, each involving fewer relations than the original query. Using both empirical and analytical methods, it is shown that generally a combination of reduction and tuple substitution will result in a lower processing cost than tuple substitution alone.

Finally, certain options are discussed which attempt to tailor the general processing algorithm to specific queries and the data they reference. These techniques take advantage of the distribution of the data and the storage characteristics of the relations. Most of these intuitively good ideas are shown to be quite useful through analysis and empirical measurements.

ACKNOWLEDGEMENTS

I am pleased to express my appreciation and deepest thanks to Prof. E. Wong for his contributions and continual support and encouragement. I am also indebted to Prof. M. Stonebraker for his helpful comments and counseling. And my thanks go to Prof. R. Fabry and Prof. M. Shanks for reading this manuscript.

Special gratitude is due to my fellow students and staff members on the INGRES project. Their effort in making this relational database system a reality and their comments and assistance were greatly appreciated.

Finally I would like to thank my husband, Kourosh, for his patience and understanding and my parents for their belief in me.

I would also like to acknowledge the research support from the Joint Services Electronics Program Contract F44620-71-C-0087.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

In the current trend of database management systems, there are three data models which are most widely used. They are the network model [CODA71], the hierarchic model [IBM70], and the relational model [CODD70]. There has been much discussion as to which of these models should be used [CODD74, DATE74, BACH74, SIBL74, HELD75, DATE75]. The advantages of the relational model have been eloquently detailed in the literature [CODD70, DATE74, CODD74, CHAM76] and hardly require further elaboration. There were two particular advantages which motivated our choice of the relational model: (1) the high degree of data independence provided, and (2) the possibility of providing a high level and entirely procedure free facility for data definition, retrieval, update, access control, support of views, and integrity verification. Such a high-level, nonprocedural language allows the system flexibility to optimize the execution of a given query and also allows for modifying the stored data structures to reflect the changing needs of the user.

The major unanswered question concerning the relational data model is whether it can be implemented efficiently. This work is concerned with the efficient processing of a user query. This includes possible transformation of the query into a more efficient statement of the request and determination of an execution plan for efficient processing of the particular query. To understand the nature of the problem, first the relational model will be formally defined and then techniques which are used in other relational systems will be discussed.

1.1 Relational Model

In mathematics, the term relation can be defined as follows: Given sets $D_1, D_2, \cdots, D_n$ (not necessarily distinct), a RELATION $R(D_1, \cdots, D_n)$ is a subset of the Cartesian product $D_1 x \cdots x D_n$. In other words, R is a set of n-tuples $X = (X_1, X_2, \cdots, X_n)$ where X_i is an element of D_i for each $i=1,2,\ldots,n$. The sets D_i are called the DOMAINS of R and R has DEGREE n. The number of tuples in R is called its CARDINALITY. The only restriction put on relations is that they be normalized [CODD71b]. Thus, each domain must be simple, that is, it cannot have members which are themselves relations.

Clearly, R can be thought of as a table with each row representing a tuple.

employee relation

name	dept	salary	manager	birth	start
Adams	candy	12000	Baker	1939	1965
Baker	admin	20000	Harding	1927	1955
Harding	admin	31000	none	1917	1949
Johnson	toy	14000	Harding	1946	1966
Jones	toy	14000	Johnson	1943	1968
Smith	toy	10000	Jones	1950	1970

Figure 1.1 A Sample Relation

An example of this representation is shown in Figure 1.1, which illustrates a relation describing employees in a department store. Observe the following properties of a relation:

1. no two rows are identical,
2. the ordering of rows is not significant,
3. the ordering of columns is significant.

Each column can be considered as a function mapping R into D_i. These functions will be called ATTRIBUTES. Note that more than one attribute can be based on the same domain.

1.2 Survey of Relational Database Management Systems

Different levels of implementation of a relational model of data can be distinguished, ranging from the user's view of an information structure to the actual storage structure of the data. Two levels are clearly distinguishable, the tuple-by-tuple access level, and the set or relation operation level. Most implementations use a low level tuple interface to the data even though the user may be unaware of this lower level system. XRM [LORI74] and the Gamma Zero interface [BJOR73] are examples of this low level interface to the relational system. Their basic purpose is to store and retrieve tuples.

User interfaces with languages based on the first-order applied predicate calculus, as introduced by Codd [CODD71], also provide a tuple access level to the data but at a higher level than, for example, XRM. Implicitly, calculus-based languages state operations on sets of tuples but these operations are stated in terms of the tuples themselves. INGRES, DAMAS [ROTH72], and System R [ASTR76] are examples of systems supporting this type of high level tuple interface, although the language used in System R is a "mapping-oriented" language rather than one based directly on the relational calculus.

Most systems which use a set or relation interface at the higher level support languages which are based on the relational algebra [CODD71c]. The operations in these languages refer to entire relations or subsets of them, thus these interfaces tend to optimize accesses of all tuples of a set at once. The MACAIMS System [GOLD70] and the PRTV System [NOTL72] are examples of implementations using sets as the basic unit of data for manipulation. (See Appendix A for definitions of the relational algebra operators.)

Codd has shown in [CODD71c] that any relational calculus expression can be reduced into a formula of the relational algebra that defines the same result relation and Palermo [PALE72] extended this algorithm by recognizing certain inefficient operations and modifying them. His improvements included performing projections prior to joins, not creating the entire cartesian product explicitly, forming the join in such an order that the result grows slowly. However, his results depend upon the assumptions that (1) a tuple is the basic retrieval unit, and (2) statistical information concerning the number of distinct values and the range of values for each domain is available. In most systems, unless the tuple size is very large or the page size very small, a tuple will not be the unit of retrieval. Also, there is considerable cost associated with gathering, maintaining and storing the required statistical information which he does not consider.

However some systems have adopted the approach of implementing the relational algebra directly. One of these is the MACAIMS system [GOLD70], developed at MIT and implemented on MULTICS. In this system, data items are encoded to a fixed-length identifier and these identifiers are used in the stored relations rather than the actual data item. Using this approach, the stored relations are usually much smaller than the corresponding relations containing actual data values and thus can be manipulated much more easily.

Another system using the relational algebra is the Peterlee Relational Test Vehicle (PRTV) [NOTL72, TODD75, TODD76] under development at the IBM Scientific Center in Peterlee, England. In this system, when a user states a query, the query is translated into a tree of operators. An optimizer then modifies the tree to reflect the system decisions concerning processing. These optimizations include rearranging and combining the algebra operators in the following manner:

1. sequences of projections on a single relation are combined into one projection.
2. sequences of selections on the same relation are combined into a single selection.
3. selections are moved as far down the tree as possible, thus allowing them to be performed earlier.
4. common subexpressions are identified and possibly evaluated.
5. removal of redundant relation operations.
6. combinations of unions and other set operators are manipulated to minimize the total size of intermediate results.

In [HALL75], these transformations are discussed in more detail and consideration is given to the order in which they should be applied. Results of experiments performed using these transformations are presented. The tree can also be utilized to represent the choice of access paths for performing specific operations.

The PRTV system additionally supports storing this tree which defines a relation so that no actual tuples are retrieved until they are needed for output. This allows for combining and simplifying a sequence of queries that keep building on the same set.

Smith and Chang [SMIT75] have independently developed similar techniques for optimizing the performance of a user query in the relational algebra. Many of the same transformations are presented but then consideration is given to which operations could run concurrently or if the information can be pipelined between two operations and to organizing intermediate results so they have the most useful sort order for the subsequent operation. Clearly, operations which must be performed on two different relations independently can be run concurrently (assuming the underlying operating system supports concurrent processes). For example, a PROJECT on the EMPLOYEE relation and a SELECT on the JOB relation can be run concurrently. There are also certain tasks which do not require a full relation to commence operating. Such tasks can be pipelined, that is, as soon as a tuple has been evaluated by the first task, it can be handed to the second task. This technique increases the throughput rate and also decreases the amount of intermediate storage required between operations. It should be noted that this pipelining cannot be done for any operation which could produce duplicate tuples since they define that each temporary relation will not contain duplicates. Also note that this type of pipelining can effectively be achieved by a single process applying all operations to the tuple. For example, a SELECT and PROJECT on the same relation can be combined into a single operation.

In addition, Smith and Chang define a set of implementation procedures for each relational algebra operator. The procedures for each operator differ in the sort order of the input relation(s) and output and thus vary in efficiency. Using these procedures and an operator tree representing the query, tasks are created "from these procedures in such a way that the performance of the whole tree of cooperating tasks is optimized. This is achieved by distributing and analyzing the effects on sort order of possible implementation decisions, and then creating tasks so as to coordinate sort order throughout the task tree." By requiring all intermediate results to contain no duplicate tuples, most often a sort will be required after each operation. It is possible that the benefits gained from this additional sort will be less than the cost of performing the sort.

Pecherer has done theoretical research on efficient operations in a relational algebra environment [PECH75, PECH75a] and also on efficient exploration of product spaces by nested iteration [PECH76]. When examining the product of arbitrarily many relations by nested iteration, different orders of iteration are possible. He compares these orders with the goal of minimizing the amount of data volume which must be transferred between secondary storage devices and main memory. Data relations are assumed to reside on secondary storage. Let n_i denote the size of relation R_i (in tuples) and b_i the number of bits per tuple of R_i. Pecherer presents the following results. At each step of the iteration, the relation R_i which maximizes the ratio $\frac{n_i b_i}{n_i - 1}$ should be selected. When only a subset of the product is to be retrieved, if the "effectiveness" of individual terms of the subsetting predicate are known and independent, an expected optimal order of iteration can be selected in a similar manner.

There has, in addition, been research on various algorithms for implementing the join operator [GOTL75, BLAS75]. Most of these algorithms exploit the use of indices or links on the join columns or provide for sorting the relations involved. No specific result of the type method A is always better was reached but it was determined that there are circumstances under which each method is best. Due to this, it is generally concluded that given the access paths available in a certain situation, an evaluator should be used to determine the method with the minimum cost.

Yet other systems use a tuple level interface based on a calculus language and opted for a more direct means of implementation than translation to a relational algebra or set interface. Rothnie [ROTH72, ROTH74, ROTH75], while at MIT, developed a system with a relational calculus language. He proposed a technique for handling a query involving two relations with a basic method which is essentially equivalent to the basic method presented here in Chapter 2. He then defines three options which can be used as extensions of the basic method. These options all use a concept of back substitution. The values of one tuple from relation A are inserted in the query resulting in a query which now involves only relation B. This query is then evaluated. The tuples of B which satisfy this new query, or the fact that no tuples satisfy, are then used to limit the remaining tuples of A which will have their values substituted into the original query. Experiments performed on certain queries illustrate that a dramatic reduction in cost can be obtained using combinations of these options [ROTH74]. However, these ideas are not easily extendable to queries involving more than two relations.

The SEQUEL system [ASTR75, ASTR75a, CHAM74], which was developed at IBM Reasearch in San Jose, provides a mapping-oriented language. Since the SEQUEL system uses XRM (Extended Relational Memory [LORI74]) as the underlying access method, the optimizer is mainly concerned with which of the XRM-supported inversions to use to limit the tuples which must be scanned for a given query. The major difficulty with this system was found to be the restrictions imposed by the block structured language. This structure limited somewhat the flexibility in selecting various orders of the relations for processing.

Another relational system, called System R, is currently under development at IBM Research in San Jose [ASTR76]. System R will support the SEQUEL language as well as other interfaces. The optimizer for this system determines a set of "reasonable" execution paths given the set of images and links which are pertinent. It then applies a cost function to determine the minimum-cost method. One of the most important parameters of the cost function is the physical clustering of tuples and this is dominant in selecting the execution method.

The optimizer of System R produces an "Optimized Package (OP)" which contains the parse tree and a plan of execution. This OP can be used directly to materialize the requested tuples or it can be saved and executed only when a specific request is made. Using this feature, it is possible to compile queries and then simply execute the OP. Thus at execution time, parsing and optimization are avoided. In a production environment, where the same query is run frequently, this can greatly reduce the processing time. However, any time that the underlying structures change, the original SEQUEL query must be reoptimized to form a new OP.

There are two relational systems under implementation at the University of Toronto, ZETA [CZAR75, MYLO75] and OMEGA [SCHM75]. Both of these systems are constructed using a multilevel architecture. In ZETA, the lowest level provides basic tuple-accessing operations, the middle level performs the interpreting and optimizing for multi-relation queries, and the top level supports several end-user interfaces. The system makes extensive use of indices [FARL75].

The OMEGA system uses an internal system language called Link and Selector Language (LSL). This is an expression-oriented language which provides subsetting operations on a relation (selector) and connections between two relations (links). The dynamic optimization supported includes choosing a fast access path, i.e., in which direction should a link be evaluated, and determining which of the existing access structures will cause the fewest data accesses. They also allow for different orders of evaluation of clauses to take advantage of inverted files which support a selector operation. This is similar to the technique used in the SEQUEL implementation [ASTR75]. Besides supporting inverted files (secondary indices) on domains of relations, they also propose a selector structure, called a subset element, which contains identifiers for tuples that satisfy a particular boolean condition. This can be a very useful tool especially when a particular subset of a relation is frequently referenced. The support for this structure is similar to that of an inverted file except that the boolean condition must be stored in some canonical form to identify the structure and the queries which contain the condition.

There is much work being done currently on relational systems but these are most of the major systems concerning the implementation of relations. There are two main areas for optimization which are considered by all of the systems. First, the order of accessing the relations to evaluate the condition on the Cartesian product, and second, making efficient use of existing access paths in this evaluation process.

1.3 Overview of Dissertation

The goal of this work is to explore ways in which the processing of a query can be performed efficiently. Most of the ideas proposed can be applied to any relational system. They will be presented with particular emphasis as to how they can be used within the INGRES system [HELD75a], since that is the system which was used for development. To this end, in Chapter 2 an introduction to the INGRES environment is provided and the query language QUEL is introduced. The implementation of INGRES is discussed in Chapter 3, in particular, the portion of the system which processes the query. Then, starting with Chapter 4, techniques for efficient processing are presented. Chapter 4 is concerned with transformations which can be applied to the query before processing begins. These are similar to techniques used in compiler optimization and those presented by Hall [HALL75] for the PRTV system. In Chapter 5, a general algorithm for decomposing a multi-relation query into a series of single relation queries which depends only on the query and not on the data structures involved is presented. Then, in Chapter 6, tailoring certain steps of this algorithm to specific queries and existing access paths is considered along with dynamic creation of new access paths. The order in which relations involved in a multi-relation query should be accessed is examined in Chapter 7. This includes a discussion of what are the critical parameters involved in the cost of processing the query. Since these ideas were developed using a working relational system, it was possible to perform some experiments to test the hypotheses of Chapters 5, 6, and 7. The results of these experiments are presented in Chapter 8. In Chapter 9, the conclusions are reviewed and areas for future work are outlined.

CHAPTER 2
INTRODUCTION TO THE INGRES ENVIRONMENT

Although the ideas presented here are, for the most part, applicable to any relational database system, it is easier to consider issues which arise in the discussion in terms of a specific system and query language. The system used for this reasearch is the INGRES system and its query language QUEL.

INGRES (INteractive Graphics and REtrieval System) is a relational database and graphics system which is implemented on top of the UNIX operating system [RITC74] developed at Bell Telephone Laboratories for Digital Equipment Corporation PDP 11/40, 11/45 and 11/70 computer systems. The implementation of INGRES is primarily programmed in "C" [RITC74a], a high level language in which UNIX itself is written. Parsing is done with the assistance of YACC, a compiler-compiler available on UNIX [JOHN74].

INGRES basically provides three user-interfaces. There is a terminal monitor which supports the primary query language, QUEL, and various utility operations. The second user interface, CUPID, is a graphics oriented, casual user language [MCDO75]. The EQUEL (Embedded QUEL) precompiler [ALLM76], which allows the substitution of a user-supplied C program for the terminal monitor, has the effect of embedding all of QUEL in the general purpose programming language "C". The utility operations currently supported are [STON76]:

1. creation and destruction of relations
2. bulk copy of data
3. modification of the storage structure of a relation
4. miscellaneous commands such as those requesting information about the database and its relations

In section 1 the query language QUEL will be described. Then, since the major topic of this research is to determine efficient ways of processing queries, a general description of the algorithm used to decompose queries will be presented in Section 2.

2.1 QUEL: A Relational Query Language

QUEL (QUEry Language) is a calculus based language and has points in common with Data Language/ALPHA [CODD71], SQUARE [BOYC74] and SEQUEL [CHAM74] in that it is a complete [CODD71c] query language which frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data. As such, it facilitates a considerable degree of data independence [STON74].

Each query of QUEL contains one or more Range-Statements and one or more Retrieve-Statements. We shall use {} to denote "one or more" and [] to denote "zero or more". With these conventions the form of a query in QUEL can be expressed as

```
    Query
      = {Range-Statement}{Retrieve-Statement}
    Range-Statement
      = RANGE OF {Variable} IS {Relation}
    Retrieve-Statement
      = RETRIEVE INTO Result-Name (Target-List)
        WHERE Qualification
    Target-List = {Result-Domain=Function}
```

The goal of a query is to create a new relation for each Retrieve-Statement. The relation so created is named by the "Result-Name" clause and the domains in that relation are named by the "Result-Domain" names given in the Target-List. In the frequent case where the Function is simply Variable.Domain-Name, the Result-Domain name may be omitted and is then taken to be the same as the Domain-Name in the Function. Also, if the "Result-Name" is TERMINAL then the result of the query is displayed on the user's terminal. To create the desired relation, first consider the product of the ranges of all variables which appear in the Target-List and the Qualification of the Retrieve-Statement. Each term in the Target-List is a function and the Qualification is a truth function, i.e., a function with values true or false, on the product space. The desired relation is created by evaluating the Target-List on the subset of the product space for which the Qualification is true, and eliminating duplicate tuples.

EXAMPLE 2.1.

CITY(CNAME, STATE, POPULATION, AREA)

"Find the population density of all cities in California with population greater than 50k"

```
RANGE OF C IS CITY
RETRIEVE INTO W(C.CNAME,DENSITY=C.POPULATION/C.AREA)
     WHERE C.STATE="California" AND C.POPULATION>50K
```

(note the default used for CNAME=C.CNAME and that the result of the query is a relation W(CNAME, DENSITY).)

It is clear from the above discussion that the basic quantities used in QUEL are functions of products of relations. The allowed functions can be exceedingly complex and fall into three categories: (a) Functions resulting from arithmetical combinations of attributes. (b) Set valued functions such as "the set of cities for each state". (c) Aggregate functions obtained by aggregating set functions, e.g., "total population of the cities of each state". The precise definition of the allowed classes of functions will be given recursively as follows: Consider a nested sequence of sublanguages of QUEL

$QUEL_0, QUEL_1, \cdots, QUEL_n, \cdots$

Let C_i denote the class of all functions and Q_i the class of all qualifications allowed in $QUEL_i$. We first define C_0 and Q_0.

C_0

(a) Any constant is in C_0.

(b) Any attribute is in C_0.

(c) If f and g are in C_0 then f+g, f-g, f*g, f/g, f**g and $\log_f g$ are in C_0.

(Note: The functions being combined need not have identical arguments. The resulting function is a function of the union of the variables.)

Q_0

(a) An atomic formula in Q_0 has the form f(comp)g, where comp is any of the comparison operators: $<$, $\leq$, $=$, $\neq$, and f and g are in C_0.

(b) Q_0 consists of all sentences made up of atomic formulas connected by the Boolean connectives: NOT, AND, OR.

Comment: A function in C_0 will be referred to as an ATTRIBUTE-FUNCTION. The value of an attribute function for a tuple depends only on the data contained in that tuple. This is not true for functions in C_i for $i>0$. A similar comment applies to Q_0 as well.

We now proceed to define $QUEL_n$ recursively. Suppose $X=(X_1,X_2,\cdots,X_m)$ are the declared tuple variables with range

$R=R_1 x R_2 x \cdots x R_m$. Let X.f and X.qual be respectively a function and a qualification allowed in $QUEL_{n-1}$. We define SET(X.f WHERE X.qual) as the set of f-values obtained by evaluating f on the subset of R for which X.qual is true, i.e.,

SET(X.f WHERE X.qual)

= {X.f: X is in R AND X.qual=true}

EXAMPLE 2.2.

X	CNAME	STATE	POPU
r1	SF	CAL	1M
r2	NYC	NY	6M
r3	CHI	ILL	4M
r4	LA	CAL	3M

SET(X.POPU WHERE X.STATE=CAL) = {1M, 3M}

SET(X.POPU WHERE X.POPU>3M) = {4M, 6M}

Comment: By definition a set contains no duplicate values. However, it is useful to define SET' as the collection obtained by retaining duplicates, for example,

SET'(X.STATE WHERE X.POPU<4M) = {CAL, CAL}

The aggregation operators COUNT, SUM, AVG, MAX, MIN, ANY have an obvious meaning when they operate on sets. If AGG is any of these operators, we shall adopt the notation

AGG(X.f WHERE X.qual)

= AGG(SET(X.f WHERE X.qual))

and AGG' will denote AGG(SET').

We shall refer to quantites of the form AGG(X.f WHERE X.qual) as AGGREGATES. An aggregate depends on the data contained in the range R but does not vary as X varies. The appearance of X merely serves to indicate the range. In this way, it acts as a dummy variable not unlike that in a definite integral. To put it more precisely, denote a function in C_n by F(X,R) to indicate the fact that in general it depends on both the tuple X and on R overall. Then we can say that constants depend on neither X nor R, functions in C_0 depend on X but not on R, aggregates depend on R but not on X.

Now suppose that f and g are in C_{n-1} and qual is in Q_{n-1}. Define

SET(X.f BY X.g WHERE X.qual)

as a set valued function of X such that it is constant on any set of X for which g is constant and on such a set it is given by

SET(X.f BY X.g WHERE X.qual)(X.g = γ)

= SET(X.f WHERE (X.g=γ) AND X.qual)

EXAMPLE 2.3.

X	X.STATE	SET(X.CNAME BY X.STATE WHERE X.POPU<5M)
r1	CAL	{SF, LA}
r2	NY	empty
r3	ILL	{CHI}
r4	CAL	{SF, LA}

The notation AGG(X.f BY X.g WHERE X.qual) is now self-explanatory, and so are the notations SET'(X.f BY X.g WHERE X.qual) and AGG'(X.f BY X.g WHERE X.qual).

EXAMPLE 2.4.

X	X.STATE	MAX(X.POPU BY X.STATE WHERE X.POPU<5M)
r1	CAL	3M
r2	NY	0
r3	ILL	4M
r4	CAL	3M

Note that AGG(X.f BY X.g WHERE X.qual), unlike aggregates, is a function of both X and R and will be called an AGGREGATE-FUNCTION. It is a function of X through X.g and only through X.g. Thus, the three appearances of X play a mixture of roles. This is an objectionable syntactic feature, which however cannot be repaired by using a dummy variable for the first and last term. AGG(X'.f BY X.g WHERE X'.qual) involves aggregation on the product of ranges of X' and X and means something quite different from AGG(X.f BY X.g WHERE X.qual). Several possible solutions have been considered but rejected for one reason or another. In particular, if X is restricted to be a single variable, then its presence can be suppressed in the first and third term. For the time being, we have chosen not to impose such a restriction.

Set functions of the form SET(X.f BY X.g WHERE X.qual) can be combined by union, intersection, and relative complement. We can define the class of set functions allowed in $QUEL_n$ as follows:

S_n

(a) S_0 contains all constant sets.

(b) S_n includes S_{n-1}.

(c) If f and g are in C_{n-1} and qual is in Q_{n-1} then SET(X.f BY X.g WHERE X.qual) and SET(X.F WHERE X.qual) are in S_n as are the corresponding SET' operations.

(d) S_n is closed under union, intersection and relative complement.

The classes C_n and Q_n can now be defined as follows:

C_n

(a) C_n includes C_{n-1}.

(b) If s is in S_n then AGG(s) and AGG'(s) are in C_n.

(c) If f and g are in C_{n-1} and qual in Q_{n-1} then AGG(X.f BY X.g WHERE X.qual) and AGG'(X.f BY X.g WHERE X.qual) are in C_n.

(d) If f and g are in C_n, then f+g, f-g, f*g, f/g, f**g and $\log_f g$ are in C_n.

Q_n

(a) Q_n contains Q_{n-1}.

(b) If f and g are in C_n, then f(comp)g is in Q_n, where comp is any of the operators $<$, $\leq$, $=$, $\neq$.

(c) If u and v are in S_n then u(set-comp)v is in Q_n where set-comp is any of the set-comparison operators inclusion, strict inclusion, equality, and inequality.

(d) If s is in S_n and γ is a value, then (γ belongs to s) is in Q_n.

(e) Q_n is closed under Boolean combinations.

EXAMPLE 2.5.

SUPPLY(SNUM, PNUM, PRICE)

Query: Find those suppliers whose price for every part that he supplies is greater than the average price for that part.

```
RANGE OF S IS SUPPLY
RETRIEVE INTO W(S.SNUM)
    WHERE   COUNT(S.PNUM   BY   S.SNUM   WHERE   S.PRICE   >
    AVG'(S.PRICE BY S.PNUM)) = COUNT(S.PNUM BY S.SNUM)
```

Comments:

(a) It is clear that the Qualification of the Retrieve-Statement is in Q_2.

(b) Instead of using COUNT, we could also have used the operator SET. In terms of processing efficiency, COUNT is preferrable.

Update statements are transformed into Retrieve-

Statements followed by a sequence of insertions and deletions. For this reason, all future examples and discussions will be in terms of Retrieve-Statements but the same reasoning will hold for updates.

2.2 Decomposition Algorithm

In database management systems, the stored data is of such large volume and of such long lifetime that it is only economical to maintain the data on low cost storage devices. Such devices will be referred to as "secondary storage" devices as opposed to the faster "main storage" which is used primarily as temporary storage during actual processing. By the nature of the data in a database system, it will almost always reside on secondary storage with portions of it being transferred to main storage for processing. The basic quantity of data transferred between main and secondary storage will be referred to as a "page". It is assumed throughout this work that the transfer of pages between main and secondary storage is costly and that by the nature of database processing, this page transfer time will be the dominant cost with actual computation time being small by comparison. Therefore, the goal of processing a query will be to examine the information in such a way so as to minimize the number of page transfers required. As long as there continues to be orders of magnitude difference in speed between main and secondary storage and as long as database processing continues to involve a high ratio of data search to computation, this will remain a valid goal.

The general algorithm used in INGRES is a uniform algorithm to deal with all queries rather than special strategies for special situations. The overall strategy can be simply stated. Rather than compiling QUEL into a lower level language, an arbitrary $QUEL_n$ query shall be decomposed into a series of one-variable $QUEL_1$ queries, at which point most of the difficult problems have disappeared. Thus, for QUEL the "optimization" which is necessary for all high level languages lies nearly entirely in decomposition.

The overall strategy has two parts: (a) A $QUEL_n$ query will be replaced by a series of $QUEL_{n-1}$ queries and one-variable $QUEL_1$ queries. (b) A multivariable $QUEL_0$ query will be decomposed into a series of one-variable $QUEL_0$ queries. Thus, repeated applications of the algorithm will decompose any $QUEL_n$ query into a series of one-variable queries in $QUEL_1$ or $QUEL_0$.

Consider a query involving one or more tuple variables $X=(X_1,X_2,\cdots,X_n)$ with range $R=R_1 x R_2 x \cdots x R_n$. Denote the qualification by Q(X) and suppose Q(X) is expanded into conjunctive normal form so that it consists of clauses connected by AND with each clause containing atomic formulas or their negation connected by OR.

(a) $QUEL_n$ -> $QUEL_{n-1}$

Suppose the query contains an aggregate function AGG(X.f BY X.g WHERE X.qual) where f and g belong to C_{n-1} and qual belongs to Q_{n-1}. Create the aggregate function and store it in a temporary relation, TEMP. Add a new variable Z with range TEMP to the query, replacing the occurrence of the aggregate function by a reference to the appropriate domain of TEMP and add any necessary linking terms to the qualification.

EXAMPLE 2.6.

```
CITY(CNAME, STATE, POPULATION, AREA)

  RANGE OF C IS CITY
  RETRIEVE INTO W(C.STATE)
      WHERE COUNT(C.CNAME BY C.STATE WHERE C.POP<4M) > 0
```

is replaced by

```
  RANGE OF C IS CITY
  RETRIEVE INTO TEMP(A=C.STATE, B=COUNT(C.CNAME BY C.STATE
  WHERE C.POP<4M))
```

and

```
  RANGE OF C,Z IS CITY,TEMP
  RETREIVE INTO W(C.STATE)
      WHERE Z.B > 0 AND C.STATE = Z.A
```

If the query contains an aggregate AGG(X.F WHERE X.qual), the result created is a single value so it is not necessary to create the temporary relation. This value simply replaces the aggregate in the query.

(b) Multivariable $QUEL_0$ -> One-Variable $QUEL_0$

(0) Stop if query is already one-variable.

(1) For each variable, say X_1 with range R_1, collect all the attributes which depend on X_1 and all the clauses in the qualification which depend only on X_1. Say $D_1, D_2, \cdots, D_k$ are the attributes, and the clauses put together yield $Q_1(X_1)$. Issue the query

```
RANGE OF $X_1$ IS $R_1$
RETRIEVE INTO $R_1'$ $(X_1.D_1, X_1.D_2, \cdots, X_1.D_k)$
    WHERE $Q_1(X_1)$
```

(2) Replace the range R_1 of X_1 in the original query by R_1'.

Comment: The purpose of (1) and (2) is to limit the range of each variable in the original query to as small a relation as possible by selecting the referenced domains and by enforcing the part of the qualification which operates only on this variable.

(3) Take the variable with the fewest tuples in its range and substitute in turn the actual values of its tuples. This reduces the number of variables by 1. After each substitution, repeat (0), (1), (2)

and (3).

Comment: Step (3) will be referred to as tuple-substitution and represents the most time consuming step in the overall algorithm.

Part (a) simply removes all aggregates and aggregate functions from the query and preprocesses them. Note that these aggregates and aggregate functions will be answered using the algorithm presented in part (b).

The algorithm presented in (b) is a general but simple method for processing any QUEL_0 query. However, there are several questions which arise when one considers tuning the algorithm for specific classes of queries.

One of the most critical steps in the algorithm is choosing which variable to tuple substitute. What are the parameters which should be considered when selecting a variable for substitution? Is the criterion of always selecting the variable whose range relation has the fewest tuples a good choice in general?

Steps (1) and (2) perform projections and restrictions over all the relations involved in the query. Intuitively, this appears to be a good strategy, but is it always so? It is possible that no reduction in size would be gained by this step or that useful structural characteristics of the relation would be lost by this operation.

Notice that steps (0), (1) and (2) are repeated for each tuple in the range of the variable selected for substitution. However, the structure of the query remaining after the substitution does not depend on the specific tuple value that is substituted. How can this fact be used to obtain more efficient processing?

Within this algorithm, tuple substitution is the only method used for reducing the number of variables. Is there another technique which can be used either in addition to or in combination with tuple substitution which will decompose a multivariable query into a series of one-variable queries?

These are the major issues which will be examined in the remainder of this work. A combination of analytical and empirical methods were used to draw the conclusions presented.

CHAPTER 3
IMPLEMENTATION

INGRES runs as a set of processes on top of the UNIX operating system. A process in UNIX is an address space (64K bytes or less for an 11/40, 128K bytes or less on an 11/45 and 11/70) which is associated with a user-id and is the unit of work scheduled by the UNIX scheduler. Processes may "fork" subprocesses. Such processes may communicate with each other via an inter-process communication facility called "pipes". A pipe is basically a one direction communication link which is written into by one process and read by another process. UNIX maintains synchronization of pipes so that no messages are lost. The INGRES processes communicate with each other via pipes.

3.1 INGRES Process Structure

Figure 3.1 shows the INGRES process structure. Process 1 is the process which communicates with the user. It may take the form of an interactive terminal monitor, an EQUEL program, or a graphics monitor for CUPID. Process 2 contains a lexical analyzer and a parser which recognize syntactically correct queries and convert them to a more convenient form for further processing. Also at this point, the qualification of the query is converted to conjunctive

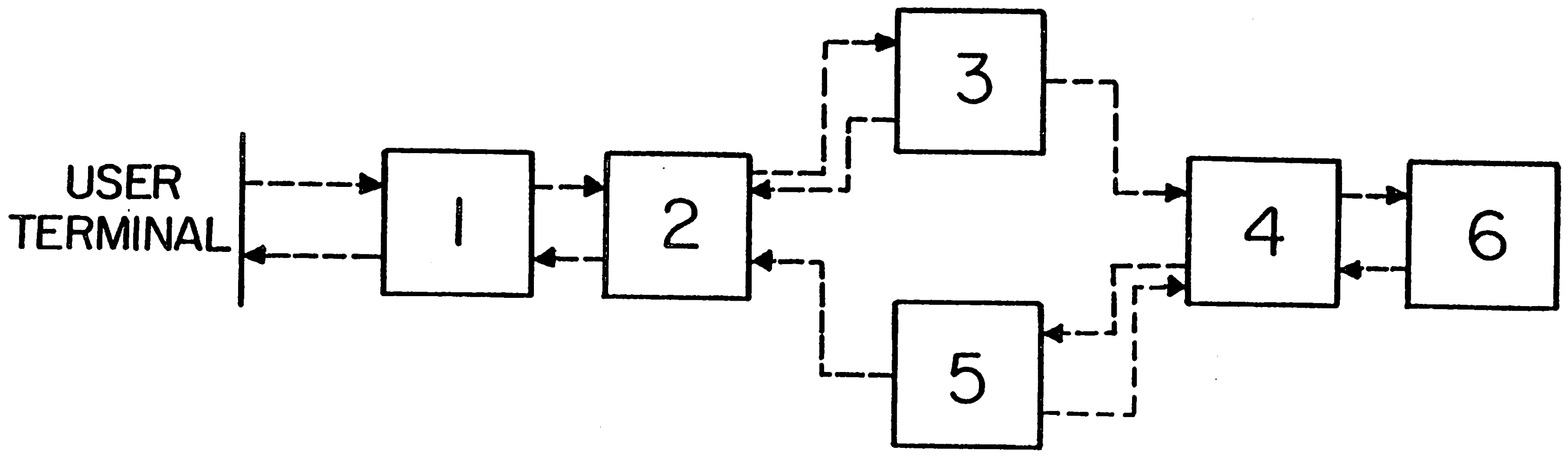


FIGURE 3.1. INGRES process structure.

normal form. The next process contains all query modification routines. Here high level protection and integrity constraints are added [STON74a, STON74b]. This process is optional and can be deleted if protection is not desired. Process 4 is where decomposition takes place. This process decomposes a query into a series of one-variable queries. These one-variable queries are then passed to Process 5, the one-variable-query-processor (OVQP), which executes them. Process 6 contains all code to support the utility commands. This process is organized as a collection of overlays which perform the various functions.

Processes 4 and 5 will be described in more detail in the following section. For a further description of any of the other processes see [STON76].

The actual accessing of data from all relations is handled through the Access Method Interface (AMI). The AMI language is implemented as a set of functions to perform the following operations:

1. open and close relations
2. get, insert, replace and delete tuples

plus other associated actions.

3.2 Decomposition Processes

In the future, Process 4 will be referred to as Decomposition and Process 5 will be called OVQP. Decomposition receives the query after all parsing and query modification has been performed. At this point, the qualification is in conjunctive normal form so that it consists of clauses connected by AND with each clause containing atomic formulas or their negation connected by OR.

The specific implementation of the general decomposition algorithm presented in Chapter 2 will now be discussed. Let Q be an aggregate-free query in variables $X_1,X_2,\cdots,X_n$ with range relations $R_1,R_2,\cdots,R_n$ respectively. Let RESULT denote the result relation for Q. The following routines will then operate on $Q(R_1,R_2,\cdots,R_n$, RESULT).

```
decomp(Q, $R_1,R_2,\cdots,R_n$, RESULT)
{
    if (Q is one-var or Q is zero-var)
          call_ovqp(Q, SOURCE, RESULT)
    else
    {
          detach_one_var(Q)
          exec_one_var($\bar{Q}_1,\cdots,\bar{Q}_n,R_1,\cdots,R_n$)
          decomp1(Q', $R'_1,\cdots,R'_n$, RESULT)
    }
}


decomp1(Q, $R_1,\cdots,R_m$, RESULT)
{
        $i$ = select_var($R_1,\ldots,R_m$)
        detach_one_var($Q_i$)
        reformat($\bar{Q}_{i1},\ldots,\bar{Q}_{im},R_1,\ldots,R_{i-1},R_{i+1},\ldots,R_m$)
        while (get_tuple($R_i$))
        {
            exec_one_var($\bar{Q}_{i1},\ldots,\bar{Q}_{im},R_1,\ldots,R_{i-1},R_{i+1},\ldots,R_m$)
            if ($Q'_i$ is one-var or $Q'_i$ is zero-var)
                call_ovqp($Q'_i$, SOURCE, RESULT)
            else
                decomp1($Q'_i,R'_1,\ldots,R'_{i-1},R'_{i+1},\ldots,R'_m$,
                RESULT)
        }
    }
```

call_ovqp(Q, SOURCE, RESULT): This routine writes the query, the name of the source and result relations and certain bookkeeping information into the pipe between Decomposition and OVQP. It then awaits a response from OVQP as to whether an error occurred or the query was answered. In the case of a user error, the error message is returned through the pipes to the user and processing of the query is terminated.

detach_one_var(Q): This routine first examines the qualification of Q. For each variable X_i, if there is a one-variable clause in X_i it is detached from Q and added to the qualification of $\overline{Q}_i$. After all one-variable clauses have been detached from Q, this results in a new query Q'. Then for each $\overline{Q}_i$ which has a non-void qualification, all domains of R_i which are referenced by Q' are included in the target list of $\overline{Q}_i$. The result of this routine is thus a new query Q' which contains no one-variable clauses and a set of queries $\{\overline{Q}_i\}$ such that each $\overline{Q}_i$ is a one-variable query in X_i.

exec_one_var($\overline{Q}_1,\ldots,\overline{Q}_k,R_1,\ldots,R_k$): This routine accepts the one-variable queries created by detach_one_var(Q) and executes them, thus resulting in a new range R'_i for each X_i for which $\overline{Q}_i$ exists. There is one exception: if the target list of $\overline{Q}_j$ is empty for some j, this means the one-variable clauses were disjoint from Q. In this case, it is not necessary to create a new range R'_j since X_j is no longer referenced by Q'. It is only necessary to verify that at least one tuple of R_j satisfies $\overline{Q}_j$. If so, processing continues as if $\overline{Q}_j$ were not present. If no tuple of R_j satisfies $\overline{Q}_j$, then no tuple satisfies Q and a null result is returned to the user.

This routine is the main routine which requires communication with Process 6, the database utilites. In order to create the new ranges, R'_i, it is necessary to write the name of the new relation and its domains to the utility process so the CREATE function can do the appropriate system bookkeeping. The relations so created are temporary relations and are later destroyed (by calling the DESTROY function in Process 6) prior to completion of the query.

select_var($R_1, \ldots, R_k$): This routine selects the variable for which tuple substitution will be performed. Currently, it compares the range sizes of all variables appearing in Q and chooses the variable whose range has the minimum number of tuples. The index of this variable is the value returned.

Since it was recognized that the criterion of minimum range size may not be the best, this routine and its usage was organized in such a way that a new criterion may be inserted without effecting the rest of the decomposition process.

reformat($\overline{Q}_1, \ldots, \overline{Q}_k, R_1, \ldots, R_k$): When a variable has been selected for substitution, a large number of queries each with one less variable will be executed. And, the structure of these queries is always the same, that is, it does not depend on the particular tuple value substituted. If, after substitution, there are one-variable clauses in some variable X_i, it is possible to modify the structure of R_i so that the domains used in $\overline{Q}_i$ are keys. This will expedite the execution of $\overline{Q}_i$ each time it is performed during tuple substitution.

The operation of reformatting involves several steps. First, if R_i is already structured such that some of the domains of $\overline{Q}_i$ are keys, it is not necessary to reformat R_i. If R_i is a small relation or $\overline{Q}_i$ will be executed only a few times, the cost of reformatting R_i can be greater than the cost of performing the query without the modify. A crude cost estimate function is included in this routine to eliminate the obviously bad cases for reformat.

Once it has been determined that R_i should be reformatted, the appropriate keys are determined by examining the qualification of $\overline{Q}_i$. Then a call is issued to the MODIFY function in Process 6 to modify the relation R_i to a hashed structure on the determined keys. (Currently, only modify to hash is supported by reformat.) These steps are peformed for each X_i for which $\overline{Q}_i$ is non-empty.

One further comment should be made. If R_i is the user's relation (i.e., not a temporary relation), then a copy is made of R_i into a temporary relation and this temporary relation is then modified. The user's relation will remain unchanged.

get_tuple(R_i): This is a call to the access method GET function which returns the next tuple of R_i into a named buffer. If there are no more tuples, a special value is returned.

That concludes the discussion of Decomposition, but since OVQP is an integral part of the decomposition process, a short description of its functions will also be presented.

OVQP is concerned solely with the efficient accessing of tuples from a single relation given a particular one-variable query. There are two major parts to this program: STRATEGY and SCAN. STRATEGY is the first step. It determines what key (if any) may be profitably used to access the relation, what the value(s) of that key will be to limit the scan of the relation, and whether the access can be accomplished on the relation directly or if a secondary index on the relation should be used. If a secondary index is to be used, then STRATEGY must determine which one of possibly many to use.

Then, SCAN processes the tuples retrieved according to the access strategy selected. This involves evaluating each tuple against the qualification of the query, creating target list values for the tuples qualifying and disposing of these tuple appropriately. At the time the qualifying typles are inserted in the result relation, it has a non-keyed structure and no checking for duplicates is performed. Thus, duplicate tuples can appear in the result relation when OVQP has finished its processing.

OVQP receives a one-variable query, the source relation and the result relation for that query, and a value which signifies whether the structure of this query is the same as the last query it processed. Note that when Decomposition gets a two-variable query, it will substitute values into the query for one of the variables and then pass the remaining one-variable query to OVQP. And, it will do this for each tuple in the first variable's range. So the query that OVQP receives will be the same each time except for certain constants. Recognizing this fact means that OVQP will not have to select an access strategy every time, it will just use the same strategy for the whole set of queries changing the appropriate limit values.

Once the query has been processed by OVQP, it returns a value as to whether the processing was successful or an error occurred. If it was successful, OVQP awaits the next one-variable query and Decomposition continues its processing. If an error occurred, an appropriate error message is returned to the user and Decomposition and OVQP both reset themselves to await a new query. Once processing of the entire user query is successfully completed, the user is informed and Decomposition and OVQP are initialized for a new query.

CHAPTER 4

TRANSFORMATIONS

Even though a query in a relational database system does not specify an algorithm for finding a result as a computer program does, there is a similarity between the two. Often what one wishes to find as the result of a query can be stated in more than one way. Although the answer does not depend on how the query is expressed, the precise expression may have an effect on how it is processed just as the statement of an algorithm within a program can have an effect on its run time. After examining some of the techniques used in compiler optimization [BAUE74, ALLE72], it was recognized that a similar methodology could be applied to queries to obtain more efficient processing. This methodology includes determining characteristics of queries which would lead to efficient processing and a means of achieving such characteristics.

The characteristics which are of interest are those which are applicable to all queries without regard to the specific data that is referenced by the query. For this reason, the techniques proposed only need to be applied once per query prior to being executed by the query processor. So these transformations will be applied after parsing and query modification have been performed but prior to decomposition.

The preprocessing proposed consists of examining each query to determine whether it possesses any characteristics which would lead to inefficient processing and, if so, applying a sequence of transformations which will produce the same result as the original query. Such transformations will provide improvement in terms of processing costs. Similar ideas have been proposed by Hall for the PRTV System [HALL75].

To determine what transformations are useful, it is necessary to have a set of goals, or desirable characteristics of queries which might serve as the target of the transformations. Such characteristics of queries are described in Section 1. In the remaining sections, the proposed transformations are categorized and discussed in some detail.

4.1 Desirable Query Characteristics

To define the characteristics of queries which will lead to more efficient processing it is necessary to understand the effect of the form in which a query is expressed upon the way in which it will be answered. It is assumed (as discussed in Chapter 2) that at the time the query is received, the qualification is in conjunctive normal form.

The following is a list of certain characteristics of queries which would be beneficial to further processing.

1. Each clause contains a minimum number of constants.
2. Each clause is in simple form, that is, it has a minimum number of operators.
3. Each clause/target list contains as few variables as possible.
4. All single relation restrictions which can be inferred from the query should be stated explicitly as clauses.
5. All constraints are consistent.

The main fact considered here is that when a one-variable query is being processed only certain clauses are used to attempt to limit the number of tuples which must be scanned. These are clauses of the form "Variable.domain op constant" where op $\in \{=, <, \leq, >, \geq\}$. By combining constants, it is likely that more clauses will be of this form and will therefore be used to reduce the amount of work involved in answering the query. For example, "E.salary/2 = 1000" is equivalent to "E.salary = 2000" and the latter form can be used more effectively. Another advantage is that if the constants are combined in this pre-processing step, this need only be done once. However, if it is done at the time the query is interpreted by OVQP, these calculations may have to be done many times as a result of tuple substitution.

One of the advantages of the second characteristic is also to decrease the amount of work in interpreting the query by OVQP. If an expression has been simplified, the number of operations to compute its value will be close to minimal. However, achieving this characteristic can also help to achieve the third characteristic, reducing the number of variables appearing in a clause. By using the laws of algebra and boolean logic it is possible to eliminate variables from a clause.

The value of the third characteristic can be realized if it is recognized that a clause involving n variables represents a condition to be verified on the n-fold cartesian product of their range relations. Therefore, each clause should contain as few variables as possible. So, if a clause can be replaced by one with fewer variables without changing the overall meaning of the query, it would be advantageous to do so. For example, X.a = Y.a AND Y.a = γ is semantically equivalent to X.a = γ AND Y.a = γ and the latter form is clearly preferable.

If a condition on an n-fold cartesian product must be verified, the relations involved should be as small as possible. Thus, if tuples which will not be of interest in the final product can be eliminated beforehand by adding a single relation restriction which is inferred by the other constraints, such an additional restrictive clause should be included in the qualification. For example, $X.a > \gamma$ AND $Y.a > X.a$ implies $Y.a > \gamma$ and this one-variable clause in Y will restrict the size of the range of Y and thus the size of the cartesian product in the ranges of X and Y which must be examined.

Any conditions which can be verified without referencing the data should obviously be performed. Usually this will require examination of several clauses together and determining their consistency with each other. For example, $X.a > \gamma$ AND $X.a < Y.a$ AND $Y.a < \gamma - 1$ considered together are inconsistent and thus the result of the query will be empty. Conditions of this form can often be recognized prior to processing the query and it is clearly advantageous to do so.

It should be realized that these characteristics are not all feasibly obtainable. In the following sections, specific transformations to achieve these characteristics are proposed and shown to preserve the overall meaning of the query. Also included is a short discussion of their value and applicability. The transformations are grouped according to the desirable query characteristic which they are trying to achieve.

4.2 Minimizing the Appearance of Constants

Transformation 2.1: combine several constants into a single constant.

$attr_1$ op β relop $attr_2$ op γ

-> $attr_1$ relop $attr_2$ op δ

where $\delta = f(\beta, \gamma)$ and op $\in \{+, -, *, /\}$

and relop $\in \{<, \leq, >, \geq, =, \neq\}$

Example:

X.a + 5 > X.b - 3 -> X.a > X.b - 8

This transformation can only be applied when all values involved are elements of the field of real numbers. Since it involves only real numbers , it is an equivalence transformation by virtue of the existence of inverses and identities for addition and multiplication in the real number system.

This manipulation can be very useful and is practical in simple cases, namely where each term in the expression is a single attribute or a constant. To apply this transformation to all possible cases would require the use of a general theorem prover to produce the resulting clause. This would be very costly in both time and space, and would, in most cases, outweigh the benefit.

If this transformation takes into account which domains the relations are keyed upon and tries to isolate such domains from the constants, this would allow much more effective use of the available access paths. However, this enhancement is not in the spirit of keeping these techniques independent of the structure of the data involved.

Transformation 2.2: if a clause contains only constants, evaluate the clause to true or false. If true, the clause can be eliminated from the query. If false, processing can be terminated and a null result returned.

This is an equivalence transformation by the truth table of the boolean operator AND, and is applicable for all values, real numbers and characters.

The relative cost of this transformation is very small and an interpreter to evaluate a constant clause already exists within the system. However, the likelihood of such a situation arising in the user's query is quite small.

Transformation 2.3: if the constant zero (0) appears within any expression, the expression can be simplified.

```
1)  expr +,- 0  ->  expr
2)  0 +,- expr  ->  (+)- expr
3)  expr * 0    ->  0
4)  expr / 0    ->  undefined  (user error)
5)  0 / expr    ->  0
```

This transformation is valid for all real numbers. 1 and 2 are equivalence transformations by the existence of identities for addition; 3 and 5 can be proven to be equivalence transformations using the identity axioms, the uniqueness property of multiplication and the distributive law; and 4 is a system defined property.

If this situation occurs, it is quite advantageous to recognize, but it is probable that this transformation will be applicable quite infrequently.

Transformation 2.4: if the constant one (1) appears as a denominator in a division or as an element of a multiplication, it can be removed.

This transformation also holds true for all real numbers and is an equivalence transformation by the existence of an identity for multiplication.

The same comment made for 2.3 holds here although the benefit of this transformation may not be as great as that of 2.3.

4.3 Simplification of Clauses

Transformation 3.1: a simple attribute divided by itself can be replaced by the constant 1.

```
X.a / X.a   ->   1
```

This conversion only works when the simple attribute represents a non-zero real number and it is an equivalence transformation by the cancellation law in the field of real numbers.

In this simple form, the transformation is nearly free, although it may not be a common occurrence. To expand it in more general terms, i.e. expr / expr -> 1, could require both a pattern matching algorithm and a theorem prover.

Transformation 3.2: reduction of a common attribute in a single clause.

X.a op_1 expr relop X.a -> 1 op_1 expr relop 1
where $op_1 \in \{*, /\}$ and X.a $\neq 0$
and relop $\in \{<, \leq, >, \geq, =, \neq\}$

Example:

X.a * Y.b > X.a -> Y.b > 1

(Comment: if X.a < 0, then the direction of the relop must be changed for the result of the transformation.)

X.a op_2 expr relop X.a -> op_2 expr relop 0
where $op_2 \in \{+, -\}$ and relop $\in \{<, \leq, >, \geq, =, \neq\}$

Example:

X.a + Y.b > X.a -> Y.b > 0

(It should be noted that the resultant clauses will be simplified further in the case where op_1=* or op_2=+.)

This transformation only holds true for values from the field of real numbers. It can be shown to be an equivalence transformation by using the inverse axioms and the cancellation laws of the real number system.

Again, this transformation is relatively inexpensive in its simple form since the query is in tree form when it is being examined. To recognize the common attribute when it appears within expr and determine if the transformation is applicable is not as easy a task. Also, if the attribute does not appear alone on one side of the relational operator, determining what is the common attribute and if it can be cancelled out becomes more difficult.

Transformation 3.3: reduction of attributes constrained to equality.

attr_1 / attr_2 relop expr AND $\text{attr}_1 = \text{attr}_2$

-> 1 relop expr AND $\text{attr}_1 = \text{attr}_2$

Example:

X.a / Y.b > Z.c AND X.a = Y.b

-> 1 > Z.c AND X.a = Y.b

This reduction is only valid for real numbers, where attr_1 and attr_2 are non-zero, and is an equivalence transformation by the identity axiom on the field of real numbers.

Since this transformation is limited to a simple form, there is little cost incurred in the recognition of it, and it can reduce the size of the cartesian product which must be examined for verification of the first clause.

Transformation 3.4: a clause which is logically negated can be simplified.

```
!(expr  relop  expr)   ->   expr  !relop  expr
        where  !relop is then simplified
        and    relop ∈ {>, ≥, <, ≤, ≠}
```

Example:

```
        !(X.a > Y.b)  ->  X.a ≤ Y.b
```

This transformation holds true for all possible values for the expressions involved and the two clauses are equivalent by virtue of the logical negation operator.

The cost of this operation is quite small. The relational operator '=' was excluded because it is of no more benefit to have expr != expr than !(expr = expr) in terms of processing. It can be added or excluded with no obvious difference.

4.4 <u>Minimization of The Number of Variables per Clause</u>

Transformation 4.1: propagate any constant values.

```
X.a  =  β   AND   expr   relop   X.a
```

-> X.a = β AND expr relop β

Example:

X.a = 5 AND Y.b = X.a -> X.a = 5 AND Y.b = 5

This transformation is valid for all values and is an equivalence transformation by the transitive property of equality.

The operation is performed very simply by examining binary maps of clauses vs. attributes. Once the one-variable clause is recognized, every other occurrence of the variable.domain combination can be replaced by the constant.

Transformation 4.2: if a variable not appearing in the target list is involved in a simple comparative clause, it can be replaced by the aggregate MIN/MAX.

expr >[<] Y.a AND Y $\notin$ tl

-> expr >[<] MIN[MAX](Y.a)

Example:

RETRIEVE (X.a) WHERE X.b > Y.c

-> RETRIEVE (X.a) WHERE X.b > MIN(Y.c)

This is a valid transformation for all values. Since the variable does not appear in the target list, it is an existentially quantified variable and it is only necessary to find the existence of one such value that satisfies the clause. So, if expr > MIN(Y.a) it is greater than at least one value of Y.a.

This operation does not directly reduce the number of variables in the clause but it effectively does. All aggregates, in this case MIN/MAX, are pre-processed by decomposition and replaced by constants. So when the original query is processed, this clause will be expr $> \beta$, where β = MIN/MAX(Y.a).

The benefit of this transformation can be realized when the cost of processing just that clause is considered. If expr contains only one other variable (say X), then before the transformation the cost is X * Y. However, after the transformation, since MIN/MAX is done separately, the cost is Y + X.

If expr contains no variables, this transformation should not be applied since it is already a one-variable (disjoint) clause which will be done prior to processing the remainder of the query.

Also, if the variable Y appears in any other multivariable clauses, this transformation should not be performed. In this case, the cost of scanning R(Y) to find the MIN/MAX is added, but R(Y) will have to be examined again to satisfy the multivariable clauses in which it appears.

If it appears in only one-variable clauses elsewhere, the transformation should be:

Y $\notin$ tl, expr > Y.a AND Y.b = β

-> Y $\notin$ tl, expr > MIN(Y.a WHERE Y.b = β)

That is, the one-variable clauses should restrict the range over which the MIN/MAX is determined.

4.5 Inferred Restrictions

Transformation 5.1: if a one-variable clause is implied by other constraints, it should be added.

X.a relop_1 β AND Y.a relop_2 X.a

-> X.a relop_1 β AND Y.a relop_2 X.a AND Y.a relop_3 β

where $\text{relop}_i \in \{<, \leq, >, \geq, =\}$

Example:

X.a > 5 AND Y.a > X.a

-> X.a > 5 AND Y.a > X.a AND Y.a > 5

The determination of relop_3 given relop_1 and relop_2 follows these rules:

1. relop_1 = '<' and relop_2 = '<', '$\leq$', '='

   -> relop_3 = '<'
2. relop_1 = '>' and relop_2 = '>', '$\geq$', '='

   -> relop_3 = '>'
3. relop_1 = '$\leq$' and relop_2 = '$\leq$', '='

   -> relop_3 = '$\leq$'
4. relop_1 = '$\leq$' and relop_2 = '<'

-> $relop_3$ = '<'

5. $relop_1$ = '$\geq$' and $relop_2$ = '$\geq$', '='

-> $relop_3$ = '$\geq$'

6. $relop_1$ = '$\geq$' and $relop_2$ = '>'

-> $relop_3$ = '>'

If $relop_1$ = '=', then this is transformation 4.1. Any other combinations of $relop_1$ and $relop_2$ do not produce valid inferences.

This is an equivalence transformation by the transitive property of ordering and is valid for all values.

This transformation is inexpensive and can greatly reduce the size of the cartesian product over which the multivariable condition must be verified. It is true that an extra clause is being added, but the only time the addition of this clause will not reduce the cost of processing is when all tuples in the relation satisfy the restriction.

4.6 Consistency of Constraints

Transformation 6.1: if two or more clauses are inconsistent, processing can be terminated and a null result returned.

Since this transformation is performed by solving a linear program, only those clauses involving real numbers, not characters, can be included in the check for inconsistency.

Consider each unique variable.domain pair as a variable in the linear program. Then the constraints can be written as:

$\sum_j a_{ij}x_j = b_i$ $\quad$ i = 1,...,no. of constraints

and $x_j \geq 0$ $\quad$ j = 1,...,no. of variables

This is the canonical form for a linear program and there are standard manipulations available which can be used to convert any linear program to this form (see [DANT63]).

The problem which needs to be solved is that of finding an initial basic solution. If there exists a feasible solution, then there are no inconsistent constraints.

Convert

$$\sum_j a_{ij}'x_j' = b_i$$

$$x_j' \geq 0$$

to

$$\sum_j a_{ij}'x_j' + y_i = b_i$$

$$x_j' \geq 0$$

$$y_i \geq 0$$

and insure that all $b_i \geq 0$.

For this set of constraints, there is an initial basic solution, namely $x_j' = 0$ for all j, and $y_i = b_i$.

So the following linear program can be solved using the simplex method.

$$\min \sum_i y_i$$

$$\text{s.t.} \quad \sum_j a_{ij}'x_j' + y_i = b_i$$

$$x_j' \geq 0$$

$$y_i \geq 0$$

There are two possibilities for the result:

$\min \sum_i y_i > 0$ => no feasible solution

=> inconsistent constraint

or

$\min \sum_i y_i = 0$ => feasible solution

=> constraints consistent

This linear program is a fairly simple one and since the number of iterations for the simplex method is usually related only to the number of constraints, it should not require many iterations to solve. Even though this is true, setting up and solving this linear program will, on the average, not be worthwhile.

CHAPTER 5

QUERY PROCESSING TECHNIQUES

The decomposition algorithm as stated in Chapter 2 is both a general and simple method for processing any query. However, it is by no means the most efficient method of processing for all queries. Within this chapter, the decomposition algorithm will be analyzed to determine if certain options can be added to make it more efficient but still allow it to be a uniform algorithm for all queries. So the ideas presented here are applicable to all queries without regard to the specific relations involved or their storage characteristics. In Section 1, the technique of tuple substitution is presented formally and possible enhancements to it are discussed. A different means of reducing the number of variables in the query, which was originally proposed by Prof. E. Wong, is called reduction and this technique will be discussed in Section 2. In Section 3, combining the methods of reduction and tuple substitution is considered as a processing strategy. Finally, in Section 4 a detailed algorithm for processing a query using our proposed policy is presented and the proposed policy is then analyzed theoretically in Section 5.

5.1 Tuple Substitution

An n-variable query implicitly requires verification of certain conditions on the n-fold cartesian product of the range relations. It is possible to examine this n-fold product on a tuple-by-tuple basis. Consider the following example.

EXAMPLE 5.1

```
RANGE OF (S,Y) IS (Supplier, Supply)
RETRIEVE (S.Sname) WHERE (Y.S#=S.S#)
```

Suppose that the range of Y is the relation

SUPPLY

S#
101
107
203

Then, successive substitution for tuples from the range of Y yields

```
Q(101):  RETRIEVE (S.Sname) WHERE (101=S.S#)

Q(107):  RETRIEVE (S.Sname) WHERE (107=S.S#)

Q(203):  RETRIEVE (S.Sname) WHERE (203=S.S#)
```

By appending the results of these three queries, the result of the original query is obtained.

This example illustrates the technique called tuple substitution. By successive substitution of a value for each variable except the last, a series of one-variable queries are generated. Each of these queries represents a portion of the cartesian product. In its simplest form, values are substituted for each variable until a one-variable query remains and then the restriction and projection are performed at the time the one-variable query is processed. In general terms, an n-variable query Q is replaced by a family of (n-1)-variable queries resulting from substituting for one of its variables tuple by tuple, i.e.

$$Q(X_1,X_2,\ldots,X_n) \rightarrow \{Q_\beta'(X_2,X_3,\ldots,X_n),\ \beta \in R_1\}$$

However, tuple substitution alone is equivalent to creating the cartesian product, so this technique simply provides a way of processing which is easier to handle on a computer, mainly a computer with limited storage. But certain steps which have beneficial results can be performed in conjunction with tuple substitution which could not be so easily done when creating the entire cartesian product at once.

5.1.1 Enhancements to Tuple Substitution

Consider first the fact that tuple substitution for a single variable means that the cost of processing the remainder of the query is multiplied by a factor equal to the cardinality of the range of the substituted variable. Therefore a logical selection procedure for choosing the substitution variable is the smallest range size first.

Now it is possible to make some of the ranges involved even smaller by performing any single variable restrictions which appear in the query prior to substitution. Consider a simple two variable query in the variables X and Y with their respective ranges, R(X) and R(Y). The cost of processing it by tuple substitution alone is the product of the two range sizes, $|R(X)|*|R(Y)|$. If there is a restriction on the range of X it can be done at a maximum cost of $|R(X)|$ (implying a complete scan of the relation), resulting in a new range R'(X). Now tuple substitution is performed over the restricted range of X at a cost of $|R'(X)|*|R(Y)|$. For the new method to do better than tuple substitution alone, $|R(X)| + |R'(X)|*|R(Y)| < |R(X)|*|R(Y)|$ which is true whenever more than a single tuple is eliminated by the restriction, assuming $|R(X)| < |R(Y)|$.

This procedure of performing restrictions before substitution can be performed at every level of substitution. And, if it is recognized that through substitution, new one-variable restrictions are created, this step can be very beneficial.

Another tool which can be used to reduce the range sizes is the elimination of unnecessary domains from a relation (selection of columns). This step can be performed in conjunction with restriction or it can be done even if a restriction over a relation is not present. Its main function, when combined with some sorting technique, is to eliminate duplicate tuple values in a single relation which would result in duplicate tuples in the final cartesian product. It also reduces the tuple width which reduces the number of pages in storage required by that relation. However, the extra cost of sorting the relation is incurred in order to determine duplicate tuple values.

5.2 Reduction

Both of the steps mentioned in 5.1.1 attempt to decrease the cost associated with tuple substitution. A different approach to the problem is a technique referred to as reduction. This method attempts to construct the result relation by assembling comparatively small pieces rather than by paring down the cartesian product. This idea takes advantage of the fact that often portions of a query "overlap" on a single variable. In general terms, a query Q is replace by Q' followed by Q" such that Q' and Q" have only a single variable in common.

Consider a query of the form

RANGE OF $(X_1, X_2, \ldots, X_n)$ IS $(R_1, R_2, \ldots, R_n)$

Q RETRIEVE $T(X_1, X_2, \ldots, X_m)$

WHERE $B''(X_1,X_2,...,X_m)$

AND $B'(X_m,X_{m+1},...,X_n)$

It is natural to break off B' to form

RANGE OF $(X_m,X_{m+1},...,X_n)$ IS $(R_m,R_{m+1},...,R_n)$

Q' RETRIEVE INTO R_m' $(T'(X_m))$

WHERE $B'(X_m,X_{m+1},...,X_n)$

where $T'(X_m)$ contains the information on X_m needed by the remainder of the query which can now be expressed as

RANGE OF $(X_1,X_2,...,X_m)$ IS $(R_1,R_2,...,R_m')$

Q" RETRIEVE $T(X_1,X_2,...,X_m)$

WHERE $B''(X_1,X_2,...,X_m)$

Note that Q" is necessarily simpler than the original query Q since $m<n$. The detachment of Q' does not lead to an increase in the maximum number of variables for which substitution has to be made. To see this, note that the maximum number of variables to be substituted for in an n-variable query is n-1. So, for Q' this number is (n-m+1)-1 and m-1 for Q" and the total is again n-1. Also, Q' and Q" are strictly ordered. Q' needs no information from Q" so that it can be processed completely before processing on Q"

begins. At any given time, it is only necessary to deal with a total of n or less variables.

There are two special cases of one overlapping-variable subqueries which should be mentioned. First, it may happen that the detached subquery Q' has no variable in common with the remainder Q". That is, B' is a function of only $(X_{m+1},\ldots,X_n)$ and not of X_m. In such a case, Q' will be called a disjoint subquery. The interpretation of this case is that if B' is satisfied by a nonempty set then Q is equivalent to Q", otherwise the result of Q is empty. The second case arises when m=n and B' is a one-variable query. This is a quite frequent and important occurrence. A query is connected if it has no disjoint subquery, one-free if it has no one-variable subquery, and irreducible if it has no one-overlapping-variable subquery. An irreducible query is obviously both connected and one-free. The variable in common between components will be called the overlapping or joining variable.

It is possible to reduce a query into components which have two or more variables in common. However the benefits recognized in the single overlapping case do not generalize to n-overlapping variables. Consider a query of the form

RANGE OF $(X_1,X_2,\ldots,X_n)$ IS $(R_1,R_2,\ldots,R_n)$

Q RETRIEVE $T(X_1,X_2,\ldots,X_m)$

WHERE B"$(X_1,X_2,\ldots,X_m)$

AND B'$(X_{m-1},X_m,\ldots,X_n)$

which breaks apart into

RANGE OF $(X_{m-1},X_m,\ldots,X_n)$ IS $(R_{m-1},R_m,\ldots,R_n)$

Q' RETRIEVE INTO R_{m-1}' (T'(X_{m-1},X_m))

WHERE B'$(X_{m-1},X_m,\ldots,X_n)$

and the remainder

RANGE OF $(X_1,X_2,\ldots,X_{m-1})$ IS $(R_1,R_2,\ldots,R_{m-1}')$

Q" RETRIEVE T$(X_1,X_2,\ldots,X_{m-1})$

WHERE B"$(X_1,X_2,\ldots,X_{m-1})$

Note that since the target list of Q' involves both X_{m-1} and X_m, X_{m-1} in Q" stands for occurrences of both X_{m-1} and X_m in Q.

The major difference is in the amount of information passed from Q' to Q". In the single overlapping case, this was only the range of a single variable. In the two overlapping case, the information needed is the (possibly restricted) cross product of the two ranges. So, even though the maximum number of variables which must be substituted for remains the same, the range of one of those variables increases. As the number of overlapping variables is increased, the number of ranges involved in this product increases. For this reason, reduction was limited to only the single overlapping variable case.

5.3 Tuple Substitution and Reduction as a Processing Alternative

It should be noted that even in the ideal case, reduction can only reduce an n-variable query to a series of two-variable queries. Since the one-variable query was chosen as the atomic unit for processing, reduction alone will not suffice. The original proposal presented in [WONG76] called for a combination of reduction and tuple substitution such that, if possible, reduction into irreducible components was always performed thus delaying tuple substitution for as long as possible. Detachment of subqueries involves an additive growth in complexity while tuple substitution involves a multiplicative growth.

After further study, it was determined that there are recognizable cases where the act of reducing will do worse than if no reduction were done. When a query is reduced, a series of component subqueries result. The only real restriction on the order of processing of those subqueries is that if two of them contain the same variable (namely the overlapping variable), one must be completely processed before the next one can be started. However there is another implied restriction. Consider the query given to illustrate single overlapping detachment. If we had chosen to process Q" before Q', the variables of Q" appear in the target list of the original query Q so they must also be passed between the two components in some manner. Either the product is passed with the range of the overlapping variable, or the product is saved and only the range of the overlapping variable is passed (a projection of this product). In the first case, the range size passed is no longer necessarily smaller than the original range of the overlapping variable, which is the same reason that two or more overlapping variables were not used. In the second case, after processing Q', it will be necessary to go back and combine the product created by Q" and the result relation of Q', which is $T(X_m)$. In either case, the cost of processing Q" followed by Q' is worse than Q' followed by Q".

So the resulting conclusion is that the component subquery which contains the original target list should be the last subquery processed. Unfortunately, this is a severe limitation to reduction and results in cases where by reducing, the actual number of tuples accessed is greater than if reduction were not performed.

EXAMPLE 5.2

Given the following three sample relations

STORE relation

number	city	state
5	San Francisco	Calif
7	El Cerrito	Calif
9	Los Angeles	Calif

SUPPLIER relation

number	name	city	state
199	Koret	Los Angeles	Calif
213	Cannon	Atlanta	Ga
33	Levi-Strauss	San Francisco	Calif
89	Fisher-Price	Boston	Mass
125	Playskool	Dallas	Tex
42	Whitman's	Denver	Colo
15	White Stag	White Plains	Neb

ITEM relation

number	name	dept	price	qoh	suppli
26	Earrings	14	1000	20	199
118	Towels, Bath	26	250	1000	213
43	Maze	49	325	200	89
106	Clock Book	49	198	150	125
23	1 lb Box	10	215	100	42
52	Jacket	60	3295	300	15
165	Jean	65	825	500	33
258	Shirt	58	650	1200	33
120	Twin Sheet	26	800	750	213
301	Boy's Jean Suit	43	1250	500	33

perform the following query.

```
RANGE OF (S,Y,I) IS (STORE, SUPPLIER, ITEM)

RETRIEVE (S.number)

WHERE  I.supplier=Y.number AND S.city=Y.city
```

where |STORE| = 3, |SUPPLIER| = 7, and |ITEM| = 10.

The processing of this query using only tuple substitution and execution of one-variable restrictions would be the following.

1. Substitute for S since it has the smallest range.
2. Perform the one-variable restriction, Y.City=β, $\beta \in$ STORE resulting in a new range R_β' for Y which depends on β.
3. Substitute for Y since it now has the smallest range.
4. Execute the remaining one-variable query in I.

For the purpose of this argument assume that the cost associated with substituting for a variable X is $|R(X)| + \sum_{\beta \in R(X)} C(Q_\beta)$. That is, the cost of reading each tuple value to be substituted plus the cost of performing the remaining query once for each substituted value. So, the cost of processing Q by the steps given above is:

$\mathrm{Cost}(Q(S,Y,I)) = 3 + \sum_{\beta \in R(S)} \mathrm{Cost}(Q_\beta(Y,I))$

$\mathrm{Cost}(Q_\beta(Y,I)) = \mathrm{Cost}(\overline{Q}_\beta(Y)) + \mathrm{Cost}(Q_\beta(Y',I))$

$\mathrm{Cost}(\overline{Q}_\beta(Y)) = 7$ for each β

for β = 1st tuple, $|R'(Y)| = 1$

$\mathrm{Cost}(Q_\beta(Y',I)) = 1 + 1*10 = 11$

for β = 2nd tuple, $|R'(Y)| = 0$

$\text{Cost}(Q_\beta(Y',I)) = 0$

for β = 3rd tuple, $|R'(Y)| = 1$

$\text{Cost}(Q_\beta(Y',I)) = 11$

So, $\text{Cost}(Q(S,Y,I)) = 3 + (7+11) + (7+0) + (7+11) = 46$

Processing this query using both reduction and substitution would involve the following steps.

1. Reduce on the joining variable Y resulting in two components, one in (Y,I) and the other in (Y,S).
2. Execute the two-variable subquery in (Y,I) since S is in the target list. This results in a new range R′ for Y.
3. Execute the two-variable query in S and Y′.

$\text{Cost}(Q(S,Y,I)) = \text{Cost}(Q(Y,I)) + \text{Cost}(Q(S,Y'))$

$$\text{Cost}(Q(Y,I)) = 7 + \sum_{\beta \in R(Y)} \text{Cost}(Q_\beta(I))$$
$$= 7 + 7*10 = 77$$

From the data it can be seen that the new range size of Y after Q(Y,I) is still 7, so

$$\text{Cost}(Q(S,Y')) = 3 + \sum_{\beta \in R(S)} \text{Cost}(Q_\beta(Y'))$$
$$= 3 + 3*7 = 24$$

Thus, $\text{Cost}(Q(S,Y,I)) = 101$.

It can easily be seen that the cost of reduction in this case is worse than if no reduction were performed.

Even though this example illustrates that reduction is not always a good idea, there are many cases where the benefits gained by reduction will be quite large. If the cases where reduction does not perform as well as substitution alone are recognizable, it is possible to use the combination of tuple substitution and reduction to determine an efficient processing strategy.

5.4 Reduction-Substitution Algorithm

The following is a detailed description of the proposed algorithm which uses both tuple substitution and reduction as processing options. The processing of a query Q has eight major steps. These steps will be presented in the overall algorithm and then each step will be discussed further.

Reduction-Substitution Algorithm

0. If Q is disjoint, reduce it. Then for each subquery, go to step 1.
1. If Q is one-variable, call OVQP (one-variable-query-processor).
2. Perform any one-variable restrictions.
3. Select a variable, X_i, for substitution.
4. Is Q reducible? If yes, go to step 5. If no, go to

step 7.

5. Should Q be reduced given X_i? If no, go to step 7, else continue.

6. Reduce Q to n subqueries, each of which is connected and one-free. For each subquery Q_j, go to step 3.

7. Substitute for X_i. For each $Q_i(\beta)$, $\beta < X_i$, go to step 1.

This algorithm is both recursive and iterative. It is recursive in the sense that after a decision has been made (steps 6 or 7), the same decision process starts over with a new subquery. It is iterative since this same recursive algorithm is applied to each subquery of substitution or reduction.

Let $X = (X_1, X_2, \cdots, X_n)$ denote the variables of Q and let T(X) and B(X) denote its target list and qualification respectively. It is assumed that B(X) is expressed in conjunctive normal form, that is

$$B(X) = \bigwedge_i C_i(X)$$

where each clause $C_i(X)$ contains only disjunctions of atomic formulas or their negation. Now consider a binary (0 or 1) matrix with p+1 rows corresponding to T(X) and the p clauses, and with n columns corresponding to the variables $X_1, \cdots, X_n$. An entry of 1 will denote the presence of a variable in a clause (or target list), and 0 will denote its absence. This matrix shall be called the <u>incidence matrix</u> of Q.

EXAMPLE 5.3

For the following query

```
RANGE OF (S,P,Y) IS (SUPPLIER, PARTS, SUPPLY)
RETRIEVE (S.sname)
WHERE S.city="New York" AND P.pname="bolt" AND P.size=20
      AND Y.snum=S.snum AND Y.pnum=P.pnum AND Y.quan≥200
```

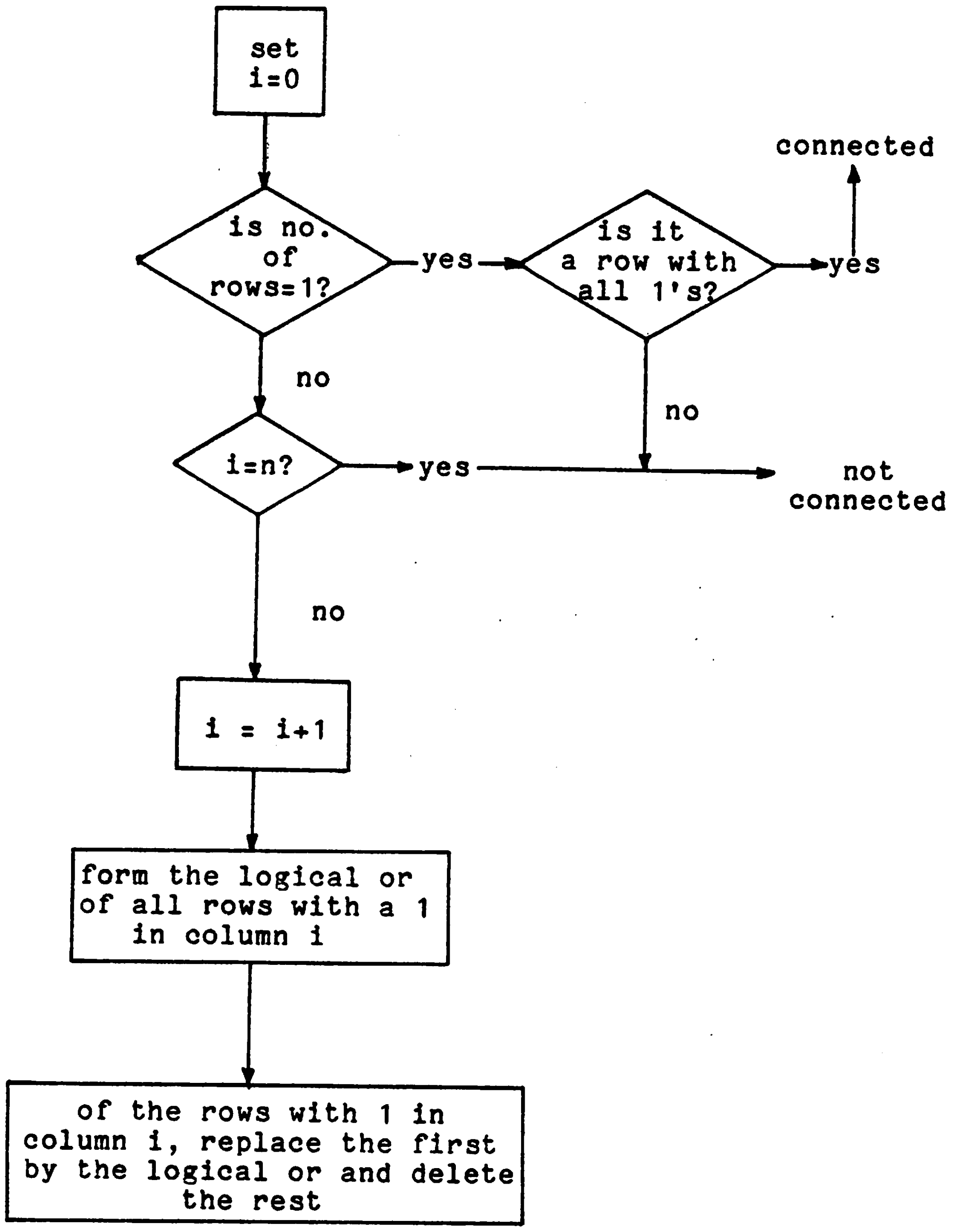
the incidence matrix is

	S	P	Y
T: S.sname	1	0	0
C_1: S.city="New York"	1	0	0
C_2: P.pname="bolt"	0	1	0
C_3: P.size=20	0	1	0
C_4: Y.snum=S.snum	1	0	1
C_5: Y.pnum=P.pnum	0	1	1
C_6: Y.quan≥200	0	0	1

For the Reduction-Substitution Algorithm there are three steps which require more detailed algorithms. Step 0 requires a test for connectedness, and a means of separating Q into disjoint components if required. Step 4 requires a means of determining if Q is reducible and Step 6 needs a way of separating Q into components and ordering them if Q is reducible.

<u>Connectivity Algorithm</u>

Figure 5.1 shows the connectivity algorithm. If the connectivity algorithm results in a matrix with a single row which is not all 1's then the variables corresponding to the zero entries are superfluous and can be eliminated. If the final matrix has more than one row, then the sets of variables corresponding to different rows must be disjoint. If the algorithm records which original rows were combined to make up each of the rows of the final matrix, then the connected components of the query can be separated.



set i=0
is no. of rows=1?
yes
is it a row with all 1's?
yes
connected
no
no
i=n?
yes
not connected
no
i = i+1
form the logical or of all rows with a 1 in column i
of the rows with 1 in column i, replace the first by the logical or and delete the rest

FIGURE 5.1. Connectivity

EXAMPLE 5.4

Consider Example 5.3 modified by the deletion of C_4. The incidence matrix now has the form

	S	P	Y
T	1	0	0
C_1	1	0	0
C_2	0	1	0
C_3	0	1	0
C_5	0	1	1
C_6	0	0	1

Applying the connectivity algorithm, we get successively

	S	P	Y
T, C_1	1	0	0
C_2	0	1	0
C_3	0	1	0
C_5	0	1	1
C_6	0	0	1

	S	P	Y
T, C_1	1	0	0
C_2, C_3, C_5	0	1	1
C_6	0	0	1

	S	P	Y
T, C_1	1	0	0
C_2, C_3, C_5, C_6	0	1	1

Hence, the query is not connected and the connected components are (T, C_1) and (C_2, C_3, C_5, C_6).

Reducibility Algorithm

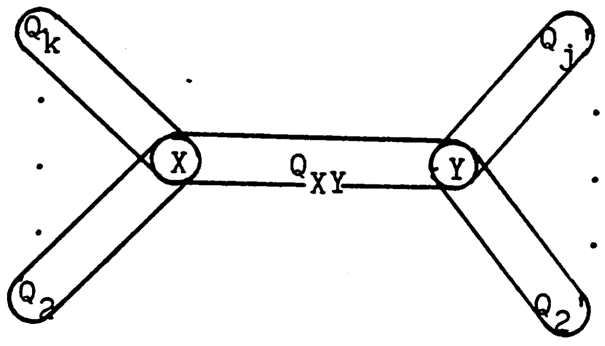
Let Q be a connected multivariable query. Observe that it is reducible if the elimination of any one variable results in Q being disconnected. Let a variable with this property be called a joining variable. Thus, Q is irreducible if and only if none of its variables is a joining variable. Joining variables have some important properties which greatly facilitate the reduction algorithm, and these are summarized as follows:

Proposition 1. Suppose that X is a joining variable of Q such that its removal disconnects Q into k connected components. Then any joining variable of one of the components is a joining variable of Q, and every joining variable of Q is a joining variable of one of the components. Further, successive elimination of two joining variables in either order results in reducing Q to the same disjoint components.

proof.

Each joining variable joins a number of components which can overlap only on the joining variable. Let X be a joining variable of Q which joins components $Q_1, Q_2, \cdots, Q_k$. Let Y be a joining variable of one of these components, say Q_1. Then, Y joins components $Q_{11}, Q_{12}, \cdots, Q_{1j}$ of Q_1, only one of which can contain X, say Q_{11}. Therefore, $(Q_{12}, \cdots, Q_{1j})$ overlaps the remainder of Q only on Y and Y is a joining variable of Q. Conversely, let Y be a joining variable of Q, and join components $Q_1', Q_2', \cdots, Q_j'$. Only one of the set $\{Q_1', \cdots, Q_j'\}$ can contain X, say Q_1', and only one of the set $\{Q_1, \cdots, Q_k\}$ can contain Y, say Q_1. Then $\{Q_2', \cdots, Q_j'\}$ and $\{Q_2, \cdots, Q_k\}$ must be disjoint since each Q_i, $i \geq 2$, can overlap its remainder in Q only on X and none of $\{Q_2', \cdots, Q_j'\}$ contains X. Hence, $Q_2', \cdots, Q_j'$ are subsets of Q_1 joined to it only by Y, so that Y is a joining variable of Q_1. It is clear that Q has components $\{Q_2, \cdots, Q_k\}$ each joined by only X, $\{Q_2', \cdots, Q_j'\}$ each joined by only Y, and a component Q_{XY} joined by both X and Y. Elimination of X and Y in either order results in disjoint components $\{\bar{Q}_2, \cdots, \bar{Q}_k, \bar{Q}_2', \cdots, \bar{Q}_j', \bar{Q}_{XY}\}$ where $\bar{Q}_i$ denotes Q_i with X removed, $\bar{Q}_i'$ denotes Q_i' with Y removed and $\bar{Q}_{XY}$ denotes Q_{XY} with both X and Y removed. Q.E.D.

The substance of Proposition 1 is illustrated by Figure 5.1.



<u>FIGURE 5.1</u>. <u>Joining Variables</u>.

The results of Proposition 1 mean that the irreducible components of Q can be found by successively checking each variable for the possibility of being a joining variable. Each variable only needs to be examined once, and the order they are tested is immaterial. Further, since a variable is joining if and only if its elimination disconnects Q, the connectivity algorithm can be used for the test.

Take the incidence matrix of Q and eliminate from it all rows with only a single "1". Beginning with the first, eliminate each column in turn and test for connectedness. Suppose that when column m is eliminated Q breaks up into k connected components with $n_1, n_2, \cdots, n_k$ variables respectively. Then, these correspond to components of Q with $n_1+1, n_2+1, \cdots, n_k+1$ variables respectively, any pair of which overlap only on X_m. Now, proceed to test columns $m+1, \ldots, n$. Note that each of the variables $X_{m+1}, \cdots, X_n$ occur in only one of the components so that after the mth column (i.e. the first joining variable) the tests are performed on matrices of reduced size.

Each irreducible component of Q corresponds to one or more rows of the incidence matrix, and can be represented by the "logical or" of the corresponding rows. Hence, Q can be represented in terms of its irreducible components by a matrix with variables as columns and components as rows. This shall be called the reduced-incidence matrix.

Section 5.5 presents a set of guidelines as to when reduction should be used rather than substitution. Using those rules and the reduced-incidence matrix of Q, it can be determined if Q should be reduced given that X_i was selected for substitution. Then it is only a simple matter of combining irreducible components to form the appropriate subqueries.

There are only two basic rules to follow in determining the order of processing for the resultant subqueries.

1) The subquery containing the original target list will be processed last.
2) If X_i is not in the subquery with the target list, the subquery containing X_i will be processed first.

Any further rules on the ordering require more information about the relations involved to predict a good ordering. The first rule above must be followed for reduction to be more efficient by the reasoning presented in Section 5.3. The second rule is not required but is logical.

Step 2 of the Reduction-Substitution Algorithm consists of detaching one-variable clauses appearing in Q and executing them. This will result in a new range size for each variable which had a restriction performed.

Step 3 of the algorithm involves selecting a variable for substitution. This step can be a very critical one in the processing algorithm and thus deserves more attention.

Chapter 7 discusses several strategies for choosing such a variable.

Step 7 performs tuple substitution. For each tuple in R_i, Q becomes an (n-1)-variable query $Q(\beta)$ in the variables $X_i,\ldots,X_{i-1},X_{i+1},\ldots,X_n$. For each β, $Q(\beta)$ is then passed to step 1 of the algorithm. The result of $Q(\beta)$ for all β in R_i are accumulated at each step.

An empirical comparison of this algorithm, using the reducibility rules presented in Section 5.5, with an algorithm using only substitution is presented in Chapter 8. Also compared is this algorithm with the option of always reducing (obviously, only if Q is reducible) to all irreducible components.

5.5 Theoretical Analysis of The Proposed Policy

The results in the previous section caused a re-examination of the policy to always reduce whenever possible. The discussion to follow assumes that it has a query and that a variable appearing in the query has been selected for substitution. The choices available are to reduce the query or to substitute for this variable. It is assumed that if reduction is performed, the same variable will be selected for substitution after the reduction. Thus, there is a continuity of the selection criterion which is unaffected by reduction. It is also assumed that all range relations are non-empty.

We introduce the following assumption.

<u>Assumption A</u>. If the substitution variable appears in the target list of the query and in a one-variable clause, at least one tuple will satisfy that one-variable clause.

The following notation will be used throughout this section. Let Q be a query in variables $X_1, X_2, \cdots, X_n$ with range relations $R_1, R_2, \cdots, R_n$. $x_i = |R_i|$, $i=1,\ldots,n$. Define $V = \{X_1, \cdots, X_n\}$ as the set of variables appearing in Q. If a variable X_i is chosen for substitution in Q, the cost of processing Q by substituting first for X_i is assumed to be $C(Q) = x_i + \sum_{\beta \in R_i} C(Q_\beta)$ which includes the cost of reading each tuple value to be substituted plus the cost of performing the remaining query once for each substituted tuple.

$C_R(Q)$ = cost of processing Q if Q is reduced

$C_S(Q)$ = cost of processing Q if substitution is performed

$C(Q(V)) = C(Q(X_1, \cdots, X_n))$

$C(Q(\emptyset)) = 0$

The following general result can be stated.

<u>Theorem 1</u>. If a query Q can be split into two disjoint subqueries (i.e. reduced into disjoint components), it will be more efficient in terms of processing cost to perform this reduction than to substitute for any variable, if

Assumption A is true.

proof.

Let Q be a query in variables $X_1, X_2, \cdots, X_n$. Define $Q_1(X_1, \cdots, X_m)$ and $Q_2(X_{m+1}, \cdots, X_n)$ as the disjoint components and let Q_2 contain the target list of Q. Let X_i be the variable chosen for substitution.

<u>Basis of induction</u>

n = 2 which implies that m = 1.

Assume $V_i = \{X_i\}$ without loss of generality.

$$C_R = C(Q_1) + \delta_1 C(Q_2) = x_1 + \delta_1 x_2$$

$$C_S = \sum_{\beta \prec R_i} (1+\gamma_\beta C(Q_\beta(V-\{X_i\}))) = x_i + (\sum_{\beta \prec R_i} \gamma_\beta) x_j$$

where

$$\delta_1 = \begin{cases} 1 & \text{if at least one tuple satisfies } Q_1 \\ 0 & \text{if no tuples satisfy } Q_1 \end{cases}$$

$$\gamma_\beta = \begin{cases} 1 & \text{if at least one tuple satisfies } Q_\beta \\ 0 & \text{if no tuples satisfy } Q_\beta \end{cases}$$

If $X_i = X_1$ then $\sum_{\beta \prec R_i} \gamma_\beta = \gamma_i x_i \geq \delta_1$ so that $C_S \geq C_R$.

If $X_i = X_2$, then by Assumption A, $\gamma_i x_i \geq 1$ which implies that $C_S \geq x_1 + x_2 \geq x_1 + \delta_1 x_2 = C_R$.

<u>Induction step</u>

n = k+1; assume theorem holds for n=k.

$$C_R = C(Q_1) + \delta_1 C(Q_2)$$

$$C_S = x_i + \sum_{\beta \prec R_i} C(Q_\beta(V-\{X_i\}))$$

where by induction

$$C(Q(V-\{X_i\})) = C(Q(X_1,\cdots,X_k)) = C(Q_1(V_1-\{X_i\}))$$
$$+ \gamma_1 C(Q_2(V_2-\{X_i\}))$$

then

$$C_S = x_i + \sum_{\beta \prec R_i} [C(Q_1(V_1-\{X_i\})) + \gamma_\beta C(Q_2(V_2-\{X_i\}))]$$

If $X_i \prec V_1$ then

$$C_S = C(Q_1) + (\sum_{\beta \prec R_i} \gamma_\beta) C(Q_2)$$

but $\gamma_i x_i \geq \delta_1$ so $C_S \geq C_R$.

If $X_i \prec V_2$ then

$$\begin{aligned} C_S &= \sum_{\beta \prec R_i} [1 + C(Q_1) + \delta_1 C(Q_2(V_2-\{X_i\}))] \\ &= x_i C(Q_1) + \delta_1 C(Q_2) + (1-\delta_1)x_i \\ &> C_R. \end{aligned}$$

Therefore, reduction into disjoint components is always better than substitution for the selected variable, under Assumption A. Q.E.D.

Assumption A is required, in the general case, because reduction is forced to do that one-variable clause as part of Q_2. If it is not satisfied, then

$$C_R = C(Q_1) + \delta_1 x_i$$

as before but now $C_S = x_i$, and the assertion $C_S > C_R$ no longer holds, which is critical in the induction step. But Assumption A does not appear as a very restrictive assumption. Note: under the current implementation of INGRES, Assumption A is not required in the induction step. Substitution for X_2 results in a constant clause which will not be interpreted until a tuple from R_1 is retrieved and it will be interpreted for each tuple of R_1. So, $C_S = x_2 + x_2 x_1$ if the one-variable clause in X_2 is not satisfied.

<u>Corollary 1</u>. Theorem 1 holds if Q reduces to n disjoint components.

This can be easily verified by noting that n-1 components can be grouped into a single component. This results in two disjoint components and Theorem 1 can then be applied. Q.E.D.

If a variable is selected for substitution and then a decision is made whether the query should be reduced at each step in the processing strategy, it is not always necessary to reduce the query to its irreducible components. If a query is split into reducible components, then a decision will be made for each of these components individually whether it should be reduced further. Due to Proposition 1 the order in which the components are reduced does not affect the final irreducible components obtained. And, by not reducing completely at each step, more flexibility in the processing strategy is gained. The following are guidelines for how much reduction should be performed given the role of the variable selected for substitution. The propositions included after these guidelines demonstrate that such "partial" reductions are possible and illustrate the means by which they can be achieved.

Let X_i denote the variable selected for substitution.

1) If X_i is a joining variable, the query will be split into components such that X_i appears in every component (that is, the query will be reduced only on X_i even if there are other joining variables in the query). Note that the resulting components are not necessarily irreducible.

2) If X_i is not a joining variable and does not appear in the irreducible component containing the target list, then the query will be split on a joining variable which will separate the query into components such that X_i and the target list are in different components and the component containing

X_i is the "smallest".

3) If X_i is not a joining variable and does appear in the irreducible component containing the target list, the query will be split on the joining variable appearing in the irreducible target list component.

All propositions make the assumption that the original query Q is connected or that the disjoint components have been detached if Q is not connected and that Q is reducible.

Proposition 2. If a variable X is not a joining variable and X is not in the irreducible component with the target list, then there is a joining variable in the irreducible component with X which will split the query into two components such that X and the target list are in different components.

proof.

The proof will be done by induction on the number of joining variables in the irreducible component containing X, IC(X).

Assume there is one joining variable in IC(X). Then if the query is split on that joining variable such that one of the resulting components is IC(X), X and the target list will be in different components by definition.

Assume that the proposition is true for n-1 joining variables in IC(X). Let there be n joining variables in IC(X).

Choose one of those, say J, and make a split on J resulting in two components, Q′ and Q" and X appears in Q". Now, either the target list is in Q′ or Q". If the target list is in Q′, then X and the target list are separated and the proof is complete. If the target list is in Q", then either

(a) J is still a joining variable of Q",

or (b) J is not a joining variable of Q".

If (a) is true, then Q" can be split again on J until (b) becomes true. This splitting process will terminate because there are only a finite number of variables in the original query and each time a split is done, the Q′ portion must contain at least one variable other than J. So there are a finite number of times that Q" can be split on J.

If (b) is true, then IC(X) now has only n-1 joining variables because J is no longer joining, so, by the induction hypothesis, the query can be split on a joining variable in IC(X) such that X and the target list are in separate components. Q.E.D.

Proposition 3. If a variable X is not a joining variable and X is not in the irreducible component with the target list, then there is one and only one joining variable in the irreducible component containing X which will separate the query into two components, one containing the target list and the other containing X.

proof.

Proposition 2 has already shown the existence of such a joining variable; this proposition will show its uniqueness.

Trivially, if there is only one joining variable in the irreducible component with X (IC(X)), this is true.

Assume there are n joining variables in IC(X), $J_1,\ldots,J_n$, such that reduction on any one of them will separate X from the target list. Split the query on all of them, $J_1,\ldots,J_n$, resulting in at least n+1 components. One of these components contains $J_1,\ldots,J_n$ and X, namely IC(X). Since each of the variables $J_1,\ldots,J_n$ separates X from the target list, the target list must appear in every component except IC(X). But, join one of these components, say the one containing J_i, back with IC(X). Then this new component and any other component have more than a single variable in common, namely the joining variable and the target list. But, by definition, any pair can have at most a single variable in common. Therefore, the target list can appear in only one component, so there is only one joining variable which will separate X from the target list. Q.E.D.

Proposition 4. If a variable X is not a joining variable and not an element of the irreducible component containing the target list, it is possible to find the "smallest" subquery containing X and not the target list. That is, it is possible to separate the query into two component subqueries Q_1 and Q_2 such that X is in Q_1 and the target list is in Q_2 and any other such split will result in X being in a component which includes Q_1.

proof.

Propositions 2 and 3 have already shown that there is always exactly one joining variable, say J^*, in the irreducible component of X which will separate X from the target list. When the query is split on J^*, two components, Q_1^* and Q_2^*, result such that Q_1^* contains the target list and Q_2^* contains X. If this separation is done so that J^* is no longer a joining variable in Q_2^*, the component Q_2^* is the "smallest" subquery containing X and not the target list. The proof of this statement will be separated into two parts.

I. Q_2^* is a subset of every other component containing X which results from a split separating X from the target list.

There are two possible ways that a separation resulting in a component other than Q_2^* can occur.

(a) there is more than one way to separate the query on J^*.

(b) there is another joining variable not in IC(X) which separates X from the target list.

These are the only possibilities since Proposition 3 states that J^* is the only joining variable in IC(X) which separates X from the target list.

Assume there is a split resulting in components Q_1 and Q_2 by (a), such that Q_1 contains the target list and Q_2 contains X. Then J^* must be a joining variable in Q_2. So Q_2 can be split on J^* until J^* is no longer a joining variable. But then this is Q_2^*, so Q_2 includes Q_2^*.

Assume there is a split resulting in components Q_1 and Q_2 by (b), such that Q_1 contains the target list and Q_2 contains X. Then J^* must also be in Q_2 since J^* is in IC(X) and X is in Q_2. But if J^* is in Q_2 it must be a joining variable in Q_2 and therefore Q_2 can be split on J^* resulting in Q_2' and Q_2'' such that X is in Q_2'' and J^* is not a joining variable in Q_2'', but then $Q_2'' = Q_2^*$. So, Q_2 includes Q_2^*.

Therefore Q_2^* is a subset of every other component containing X which results from a split separating X from the target list.

II. This is the "smallest" separation, that is, there is no other separation which results in a component containing X which is a subset of Q_2^*.

Assume there is such a component, Q_2, resulting from a split on J_1. J_1 must be a joining variable in the irreducible component containing X otherwise Q_2 includes Q_2^* by part I. But, by Proposition 3, then $J_1 = J^*$ because there is only one joining variable in IC(X) which separates X from the target list. Therefore, Q_2^* is the smallest.

Thus, Q_2^* is a subset of every other component containing X and not the target list and there is no separation which will result in a component smaller than Q_2^*. Q.E.D.

After spending a great deal of time comparing the cost of processing queries using reduction or substitution on a case by case basis, it was determined that the structure of the query was the critical parameter and a means of representing this structure would be most helpful. Several unsuccessful attempts were made to find a graphical representation to represent both the structural characteristics and the cost associated with that structure. The graphical representation which was selected for use to aid the anaylsis depicts only the structure of the query. But, using this representation and a specific cost function, it was possible to partition the queries into several groups by recognizing patterns in the graph. This graphical representation will now be defined and several examples given to illustrate its use.

Let Q be a query with variables $X_1,X_2,\cdots,X_n$. Denote $X=X_1,\cdots,X_n$. Let T(X) be the target list of Q and let B(X) be the qualification of Q. We assume that B(X) is in conjunctive normal form, i.e.,

$$B(X) = \bigwedge_i C_i(X)$$

where $C_i(X)$ represents a clause which contains only disjunctions of atomic formulas or their negation.

Define a graph $G(N,E)$ to represent Q such that

$N = \{X_1, X_2, \cdots, X_n\}$ = set of nodes

E_D = set of direct edges

nodes i, j are connected directly if variables i and j appear in clause C_k such that $|C_k| = 2$. This will be represented by a solid line between nodes i and j.

E_{ID} = set of indirect edges

nodes i and j are connected indirectly if variables i and j appear in clause C_k such that $|C_k| > 2$. This will be represented by a dotted line between nodes i and j.

$E_E = E_D \cup E_{ID}$ = explicit edges

E_I = implicit edges

nodes i and j are connected implicitly if variables i and j appear in T(X). If variable $i \in T(X)$, node i will be circled on the graph. All circled nodes are implicitly connected but no actual edge will be drawn on the graph.

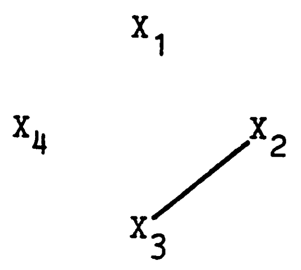
$E = E_E \cup E_I$

EXAMPLE 5.5.

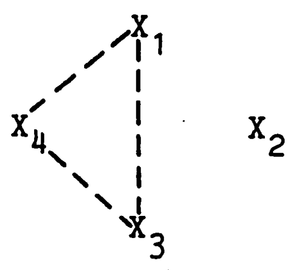
Q: RETRIEVE (X_1.a, X_2.a)
WHERE X_2.b=X_3.b AND X_1.c=X_3.c+X_4.c

N = $\{X_1,X_2,X_3,X_4\}$

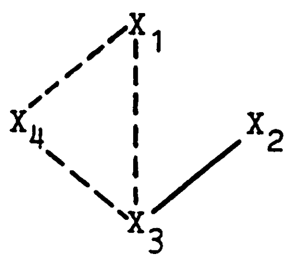
G(N,E_D)



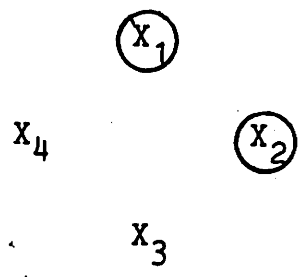
G(N,E_{ID})



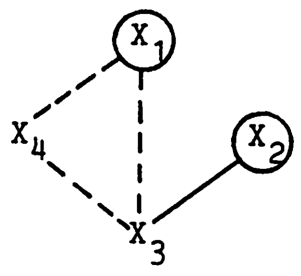
G(N,E_E)



G(N,E_I)



G(N,E)



A query Q is <u>completely disconnected</u> if the graph G(N,E) is disconnected.

A query Q is <u>explicitly disconnected</u> if the graph G(N,E_E) is disconnected.

A completely disconnected query is explicitly disconnected.

Given a graph G(N,E) for a query Q and a variable X_i to be substituted, define $G_S(N_i, E_E \cap (N_i X N_i))$ as the graph of Q if X_i is substituted tuple by tuple where $N_i = N - X_i$. This involves the following operations on G.

1) remove the node X_i from G(N,E).
2) remove all explicit edges from G(N,E) which were connected to the node X_i.

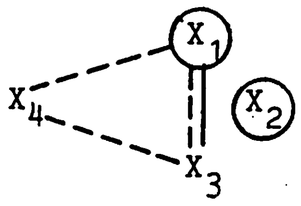
Define $G_R = \{G_j\}$ as the graph of Q if the query is reduced given that X_i was selected for substitution.

1) reduce Q on the appropriate joining variable, X_j, resulting in n subqueries.
2) create a graph G_j for each of the n subqueries. The node circled (i.e. the variable appearing in the target list) in graphs $G_1, \cdots, G_{n-1}$ will be only X_j with the understanding that Q_n corresponding to G_n contains the original target list of Q.

EXAMPLE 5.6.

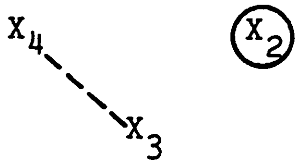
```
Q:    RETRIEVE ($X_1$.a, $X_2$.a)
      WHERE $X_1$.b=$X_3$.b AND
                ($X_1$.d=$X_4$.d OR $X_3$.c=$X_4$.c)
```

G(N,E)



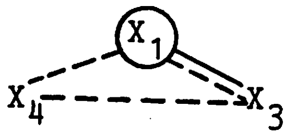
X_1 is the substitution variable.

G_S $Q_S(X_1)$ is completely disconnected

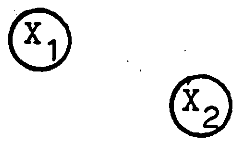


$G_R = \{G_1, G_2\}$ $X_j = X_1$

G_1

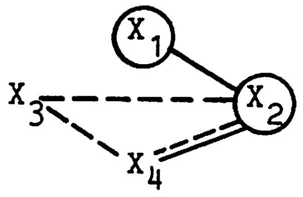


G_2



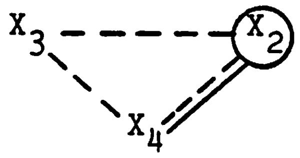
Q: RETRIEVE (X_1.a, X_2.a)
WHERE X_2.b=X_4.b AND X_1.b=X_2.d
AND X_2.c=X_3.c+X_4.d

G(N,E)



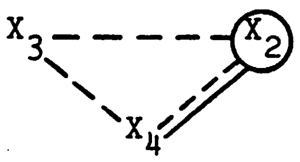
X_1 is the substitution variable.

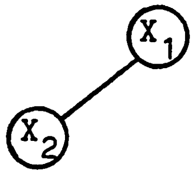
G_S



G_R = {G_1, G_2} $X_j = X_2$

G_1 G_2





The following theorem and conjectures are a result of the analyses performed, using the above graphical representation of a query and the cost function as described at the beginning of this section.

It is assumed that the original query Q has no disjoint subqueries, since by Theorem 1, these should be reduced. It is also assumed that any time an operation results in disjoint subqueries, this fact will be taken advantage of. Obviously, all queries under consideration will be assumed reducible.

<u>Theorem</u> <u>2</u>. Let X_i be the variable selected for substitution. If X_i is a joining variable, then Q_S will be completely disconnected and the cost of substituting for X_i will be less than the cost of reduction followed by substitution for X_i first in each of the resulting components.

proof.

If X_i is a joining variable, then by definition it is the only variable in common among two or more subqueries. Therefore, by substituting for X_i, these two or more subqueries will be disjoint.

Let n = number of connected components in G_S

= number of subqueries in G_R

<u>Induction basis</u>

n=2

$$C_S = \sum_{\beta \in R_i} (1 + C(Q_1) + \gamma_\beta C(Q_2))$$
$$= x_i + x_i C(Q_1) + \sum_{\beta \in R_i} \gamma_\beta C(Q_2)$$

where $\gamma_\beta = \begin{cases} 0 & \text{if } Q_{1\beta} \text{ is not satisfied} \\ 1 & \text{if } Q_{1\beta} \text{ is satisfied} \end{cases}$

$$C_R = C(Q_1(V_1 \cup \{X_i\})) + \delta_1 C(Q_2(V_2 \cup \{X_i^1\}))$$
$$= \sum_{\beta \in R_i} (1+C(Q_1)) + \delta_1 \sum_{\beta \in R_i^1} (1+C(Q_2))$$
$$= x_i + x_i C(Q_1) + \delta_1 x_i^1 + \delta_1 x_i^1 C(Q_2)$$

where R_i^1 = the relation carried over between subqueries which is = {tuples $\in R_i$: $\gamma_\beta = 1$} (A).

By (A)

$\delta_1 x_i^1 = \sum_{\beta < R_i} \gamma_\beta$ so $C_R = C_S + \delta_1 x_i^1 \geq C_S$.

<u>Induction step</u>

Assume the theorem is true for n-1, prove for n.

$$C_R = C(Q_1 U Q_2 U \cdots U Q_{n-1}) + \delta_{n-1} C(Q_n(V_n U\{X_i^{n-1}\}))$$

$$\geq C_S(Q_1 U \cdots U Q_{n-1}) + \delta_{n-1} C(Q_n(V_n U\{X_i^{n-1}\}))$$

by the induction hypothesis

$$= C_S(Q_1 U \cdots U Q_{n-1}) + \delta_{n-1} x_i^{n-1}(1+C(Q_n))$$

$$C_S = \sum_{\beta < R_i} (1 + C(Q_1) + \gamma_\beta^1 C(Q_2) + \cdots + \gamma_\beta^{n-2} C(Q_{n-1}) + \gamma_\beta^{n-1} C(Q_n))$$

$$= C_S(Q_1 U \cdots U Q_{n-1}) + [\sum_{\beta < R_i} \gamma_\beta^{n-1}] C(Q_n)$$

But $\sum_{\beta < R_i} \gamma_\beta^{n-1} = \delta_{n-1} x_i^{n-1}$ by the same reasoning as (A) applied n-2 times.

So,

$$C_S = C_S(Q_1 U \cdots U Q_{n-1}) + \delta_{n-1} x_i^{n-1} C(Q_n).$$

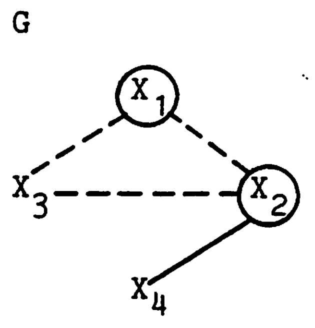
Therefore, $C_R \geq C_S + \delta_{n-1} x_i^{n-1} \geq C_S$. Q.E.D.

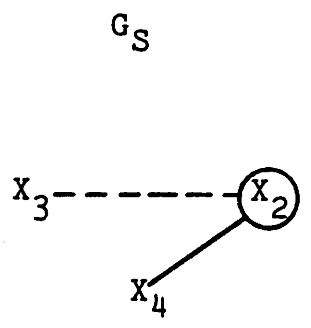
The following results are presented as conjectures. They are believed to be true but it was not possible to develop a general proof using the methodology presented above.

<u>Conjecture 1</u>. Let Q be a query and X_1 be the variable chosen for substitution. Assume $X_1 \nless T(X)$ and if $X_1 \nless B(X)$ then $X_1 \nless C_k$ such that C_k is a three or more variable clause (i.e., X_1 has no direct connections to any other nodes in G). Then, if G_S is connected, reduction is a better strategy than substitution in terms of processing cost.

<u>Reasoning</u>. In this case, substitution for X_1 basically loses because this substitution will have no immediate effect on the query. It just implies that the cost of the remaining query will be multiplied by $|R_1|$ and no other range relations will be affected. Reduction, by splitting the cost of the whole query into a sum of costs for smaller queries, has a good chance of being less than the substitution cost.

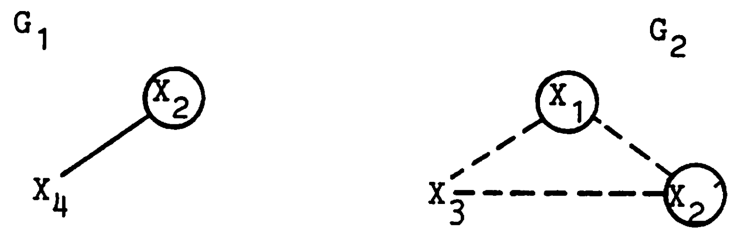
EXAMPLE.





$$C_S = x_1(1+x_2(1+x_4+px_4x_3))$$

G_R:

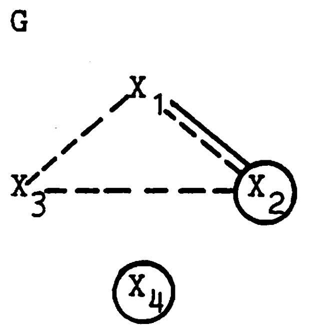


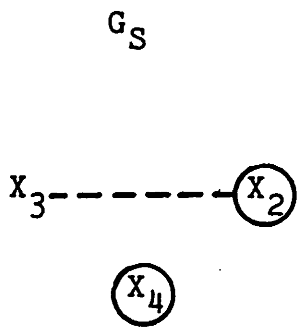
$$C_R = x_2(1+x_4) + x_1(1+x_2'(1+x_3))$$

<u>Conjecture</u> <u>2</u>. Let Q be a query and X_1 the variable chosen for substitution. Assume $X_1 \nless T(X)$. then if X_1 is directly connected to some node $X_i < T(X)$ and G_S is explicitly disconnected, substitution will do better than reduction in terms of processing cost.

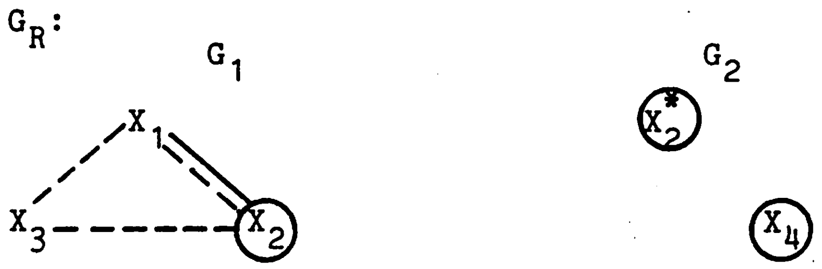
<u>Reasoning</u>. This situation implies that X_i is the joining variable for reduction, otherwise X_1 must be joining and this case is covered by Theorem 2. Therefore, X_1 will appear in the first component of reduction and reduction will match the substitution cost except that reduction must re-read the range of X_i.

EXAMPLE.





$$C_S = x_1(1+x_2+x_2'(1+x_3+px_3x_4))$$



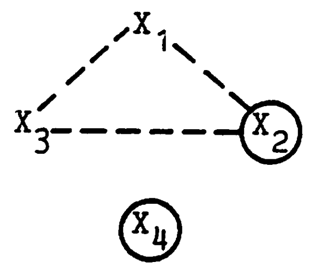
$$C_R = x_1(1+x_2+x_2'(1+x_3)) + x_2^*(1+x_4)$$

<u>Conjecture 3</u>. Let Q be a query and let X_1 be the variable chosen for substitution. If X_1 has no direct connections to any node $X_i \leqslant T(X)$ and G_S is explicitly disconnected, then reduction will do better than substitution in terms of processing cost.

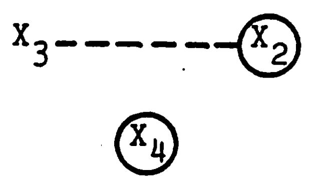
<u>Reasoning</u>. Here the fact that X_i is joining and gets restricted before executing the second component is the winning point for reduction. Substitution must substitute for at least two variables before it will get the same effect.

EXAMPLE.

G



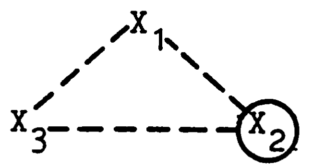
G_S



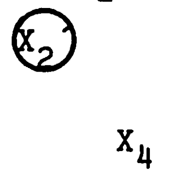
$$C_S = x_1(1+x_2(1+x_3+px_3x_4))$$

G_R:

G_1



G_2



$$C_R = x_1(1+x_2(1+x_3)) + x_2'(1+x_4)$$

Conclusions

As a result of the previous discussion, it is proposed that in the following cases substitution for the selected variable should be performed:

1) when the variable selected for substitution is a joining variable,
2) when the variable selected for substitution is not in the target list but is involved in a two-variable clause with some other variable that is in the target list.

In all other cases reduction should be performed on an appropriate joining variable. It is true that the previous discussion does not show that reduction is the winning strategy for all other cases and, depending on the distribution of the domains, substitution could be less expensive in certain cases. However, it is felt that generally reduction has a much better chance of incurring a lesser cost than substitution. This is particularly true if an operation to remove duplicates in the result relation is included between the processing of subsequent components resulting from reduction.

CHAPTER 6

DATA DEPENDENT OPTIMIZATION TECHNIQUES

In the previous chapters, techniques for efficient processing which did not depend on the characteristics or structure of the data involved were discussed. However, if certain information is known about the data itself there are many ways to use this knowledge effectively. The information which can be of use falls into two catagories, that pertaining to the distribution of values within domains and that pertaining to the way the data is actually stored within the system.

Distributional information can be very useful when answering queries involving more than one relation. These queries require searching multiple relations and the order in which these searches are performed can greatly affect the processing costs. Using the distribution, it is possible to estimate the cost of different orderings and thus eliminate at least the obviously expensive ones. Also, at one point in the algorithm, all single relation restrictions appearing in the qualification are executed. If the distributional information is available, it would be possible to determine which of these restrictions would do the most good and which would do no good at all in terms of reducing the number of tuples which must be retained for resolution of the remaining query. Those restrictions resulting in the selection of the fewest tuples can be done initially in the possibly remote hope that no tuples will satisfy and processing beyond that point will be unnecessary. The restrictions which would select a high percentage of the tuples can be put off indefinitely and perhaps not performed until that relation must be examined to satisfy some other condition.

The second category of information is concerned with how the data is stored and the access paths that are available to it. There are two general classes of data structures, keyed and non-keyed. A keyed structure is one in which a domain (or combination of domains) of a tuple is used to determine where in secondary storage the tuple should be stored. This domain is called the "primary key" or simply, "key". In such structures, when a value of the key domain is specified, the tuple(s) having the specified value can be located directly without a full scan of the relation. On the other hand, non-keyed structures are ones in which the tuples are stored using some criteria which does not depend on the value of the tuple. Non-keyed structures do not provide any ability to limit the number of tuples examined when specific values of one or more domains are supplied.

Knowing whether a relation is stored as a keyed or non-keyed structure, what domains are keys and if there are any secondary indices (inversions) on the relation all determine how the data can be accessed. Compiling this information and determining an efficient access path for a given single relation query is not a straight forward process. However, that is the problem addressed by the strategy portion of OVQP and is not discussed here. But, if decomposition is aware of these considerations and uses them in determining its overall strategy, many benefits can be gained. For example, before passing a one-variable query to OVQP, decomposition can restructure the relation involved so that it has a useful access path. Or, in determining which variable to tuple-substitute, it can select one such that the remaining relation can be efficiently accessed by OVQP.

These two types of data-dependent information can be used together at each step in the processing of a query to determine an efficient means for answering the query. The information upon the storage characteristics is readily available and incurs no extra cost since it is required by the access methods to retrieve the stored data. However, the information concerning the distribution of values for each domain is not required by any other part of the system and there could be considerable cost associated with compiling and maintaining it. Before requiring the use of this distributional information in the processing decisions of decomposition, it is necessary to consider the steps when it could be used and if the benefits gained from its usage will outweigh the associated costs. The purpose of this chapter is to discuss the processing steps which could take advantage of the distributional information and to analyze the effect it would have on the decisions made at each step.

In the first section an introduction to the analysis is presented. This includes a discussion of the model used for estimating costs and the assumptions and terminology which will be used throughout the chapter. In the next four sections, the use of statistics in the following processing steps is analyzed: (1) preprocessing one-variable restrictions, (2) projecting a relation prior to tuple substituting for it, (3) reformatting (modifying the storage structure) of remaining relations taking into account the effect of the substitution which just occurred, and (4) creating a secondary index rather than reformatting. Then, in Section 6.6, the cost and effectiveness of various types of distributional information which could be used are compared. In this chapter, the analysis of these options is presented; the results of empirical studies of the same questions are presented in Chapter 8.

It should be noted that the results of this chapter will not be a definite answer as to whether distributional information should be maintained. These analyses are presented simply to examine the effect statistics would have on these particular decisions. And the steps discussed here are not the only places where distributional information could be used. One major area where these statistics would be invaluable is when trying to estimate the costs of various processing paths. This problem is discussed in Chapter 7.

6.1 Introduction to the Analysis

Since the decision made at each step should reflect the option with the minimum processing cost, it is necessary to develop a cost function to compare the decision which would be made when statistics are available and the choice without statistics. Let Q be a query in variables $X_1,X_2,\cdots,X_n$ with respective ranges $R_1,R_2,\cdots,R_n$. The following terminology will be used throughout the analyses.

t_i	the number of tuples in relation R_i.
w_i	the width of a tuple in relation R_i in bytes.
w_i'	the width after referenced columns have been selected from R_i.
c_i	tuple capacity of a page in R_i - the number of tuples in R_i which can fit on a single page of secondary storage (the integer portion of page size divided by tuple size since INGRES does not allow a tuple to be split between pages).
c_i'	tuple capacity of a page in R_i where tuple width

is w_i'.

P_i the number of pages required to store relation R_i.

The access methods currently provide two page buffers for use in accessing relations. This fact means that when OVQP is reading tuples from the source relation and writing tuples to the result relation, a page from each relation will remain in main storage so that the cost of OVQP's processing is essentially in pages even though it works on a tuple at a time. However, this fact does not aid decomposition while it is tuple substituting. After a tuple is read and its value substituted, if any other operations are performed, such as opening a temporary relation which is the result of a one-variable restriction, before going back to get the next tuple value for substitution, the page will have to be re-read. So, decomposition has to read a page for every tuple in most cases.

These observations lead to the cost of processing Q, in pages referenced, using only tuple substitution and assuming that the order of substitution is $X_1, X_2, \cdots, X_n$ as

$$C = t_1 + t_1(t_2 + t_2(t_3 + \cdots + t_{n-1}(P_n) \cdots)$$

The first term is the cost of reading each tuple of R_1 to substitute its value; the second term is the cost of reading each tuple of R_2 once for each tuple of R_1; etc. The final term is the cost of performing the one-variable query in X_n

once for every combination of tuples from relations $R_1,\dots,R_{n-1}$. This formulation assumes that R_n does not have a useful keyed structure which OVQP can use to limit the scan, thus it must access all pages.

If Q contains a one-variable clause in X_1 which is to be executed prior to substitution of X_1, this will be reflected in the cost function by

$$C = P_1 + t_1' + t_1'(t_2+t_2(t_3+\dots+t_{n-1}(P_n)\dots)$$

P_1 is the cost for OVQP to perform the one-variable restriction in X_1 (again assuming all pages must be accessed), resulting in t_1' tuples in the new range R_1'.

Throughout the analyses to follow these assumptions will be made:

(a) unless specifically stated otherwise, the relation being accessed in any one-variable query has no useful keyed structure so that OVQP will have to perform a full scan of the relation.

(b) $t_i \geq 1$ for all $i=1,\dots,n$.

(c) tuple substitution is the only method available for reducing the number of variables. Since the options discussed either refer specifically to substitution or play the same role in reduction as in substitution, this assumption will have no effect on the generality of the results.

All costs will be in units of pages referenced.

6.2 One-Variable Restrictions

Consider the following two variable query

FIND THE PART NAMES AND SUPPLIER NUMBERS FOR ALL PARTS USED BY PROJECT 10 WHICH HAVE LESS THAN 10 UNITS ON HAND

```
     RANGE OF (P, S) IS (PARTS, SUPPLY)
     RETRIEVE (P.pname, S.snum)  WHERE  P.pnum = S.pnum
                 AND P.qoh < 10 AND S.jnum = 10
```

Since this is a two variable query, reduction is not an available tactic and the remaining alternatives are tuple substitution and detachment of one-variable subqueries. The question being addressed here is whether the one-variable clauses should be processed before tuple substitution commences.

Intuitively, performing all restrictions prior to substitution is a good idea since the cost of processing a one-variable query is small compared to the possible reduction in the size of the cross product which must be examined. Consider a three-variable query in X_1, X_2, X_3. If the cross product was constructed by tuple substitution alone, the cost would be

$$C = t_1 + t_1 t_2 + t_1 t_2 P_3$$

The dominant term in this cost is clearly $t_1 t_2 P_3$ (assuming

t_1, t_2, P_3 reasonably large). If this term can be reduced by simply adding a linear term to the cost, namely the cost of performing a restriction, then it would appear advantageous to do so. For example, if a restriction in X_2 was present, the cost would become

$$C = P_2 + t_1 + t_1 t_2' + t_1 t_2' P_3$$

And, hopefully $t_2' << t_2$. This example illustrates the idea behind performing one-variable restrictions, that is, to reduce the size of the cross product which must be examined.

6.2.1 Generalized Restriction Problem

For the purpose of this discussion it will be assumed that an n-variable query has the following characteristics:

(1) one or more clauses involving all n variables,

(2) at least one one-variable restrictive clause for each of the n variables.

And all clauses are of type (1) or (2).

This assumption is made to simplify the calculations which are presented and does not affect the generality of the results. If there is a clause with $m<n$ variables, then at some point during the substitution it will become a one-variable clause. At that point the query will have characteristics (1) and (2) and the analysis can be applied. By making this assumption, it allows the cost function to

reflect the cost of performing these restrictions all at once prior to any substitution. Otherwise it must be determined what clauses become one-variable after which level of substitution. Characteristic (2) can be relaxed so that only $m<n$ variables have one-variable clauses present but in its current form it allows for the general analysis.

Let p_i, $i=1,2,\ldots,n$, be the percentage of tuples (pages) in R_i which satisfy the corresponding one-variable restriction, $0<p_i\leq 1$ (p_i can be zero, but obviously in this case, performing the restriction is the correct action). Let C_i, $i=0,1,\ldots,n$, be the cost of processing Q given that i of the available n restrictions are performed prior to substitution.

The results will assume, for simplicity, that the order of substitution will be $X_1,X_2,\cdots,X_n$. In the general form,

$$C_m = \sum_{i=1}^{m} P_i + p_1t_1(1+p_2t_2(1+\cdots+p_mt_m(1+t_{m+1}(1+\cdots+t_{n-1}(P_n)\cdots)$$

For all restrictions to be performed, it must be true that $C_n\leq C_{n-1}\leq\cdots\leq C_0$. The conditions forced upon the p_i's by this ordering will be examined by considering pairs of the C_i's.

$C_1 \leq C_0$ if and only if

$$p_1 \leq 1 - \frac{P_1}{t_1(1+t_2+t_2t_3+\cdots+t_2t_3\cdots t_{n-1}P_n)} = \bar{p}_1$$

$C_2 \leq C_1$ if and only if

$$p_2 \leq 1 - \frac{P_2}{p_1 t_1 t_2 (1+t_3+t_3t_4+\cdots+t_3t_4\cdots t_{n-1}P_n)} = \bar{p}_2$$

$C_m \leq C_{m-1}$ if and only if

$$p_m \leq 1 - \frac{P_m}{p_1 t_1 \cdots p_{m-1} t_{m-1} t_m (1+t_{m+1}+\cdots+t_{m+1}\cdots t_{n-1}P_n)} = \bar{p}_m$$

$C_n \leq C_{n-1}$ if and only if

$$p_n \leq 1 - \frac{1}{p_1 t_1 p_2 t_2 \cdots p_{n-1} t_{n-1}} = \bar{p}_n$$

For C_n to be the minimum cost, all of these conditions must be satisfied, that is $p_i \leq \bar{p}_i$ for i=1,...,n. Clearly, if $P_1 \leq P_2 \leq \cdots \leq P_{n-1}$ and $P_n > 0$, the $\bar{p}_i$'s form a decreasing sequence, i.e., $\bar{p}_i < \bar{p}_{i-1}$. So, if $p_i \leq \bar{p}_n$, i=1,...,n, C_n will be the minimum cost. But,

$$\bar{p}_n = \frac{\prod_{i=1}^{n-1} p_i t_i - 1}{\prod_{i=1}^{n-1} p_i t_i}$$

which in most cases will be very close to 1. If no ordering is required on the P_i's, each bound can be examined individually remembering that $\frac{P_i}{t_i} \leq 1$. If the t_i's are large, the sum of products term in the denominator of each $\bar{p}_i$ will be quite large, thus bringing the bound closer to 1. Examining the $\bar{p}_n$ expression, note that if any two of the $p_i t_i$ terms are greater than 3 (tuples), $\bar{p}_n \geq .9$.

This analysis illustrates that even if the exact p_i's were known for each one-variable restriction, in the majority of the cases the restriction should be performed due to the fact that the bounds on the p_i's beyond which no benefit would be gained are so close to 1. So, for the purpose of deciding whether to perform the restrictions, distributional information will not have a major effect.

6.3. Projection Prior to Substitution

Consider the following two-variable query

```
FIND THOSE PARTS WHICH ARE SUPPLIED BY SOME SUPPLIER
     RANGE OF (P, S) IS (PARTS, SUPPLY)
     RETRIEVE (P.pname)  WHERE  P.pnum = S.pnum
```

Suppose that S was selected for tuple substitution; then the remaining one-variable query in P would be executed $|R(S)|$ times. Now, it is possible that more than one supplier supplies parts with the same part number so there could be duplicate values in the pnum column of SUPPLY. If only the column of SUPPLY representing pnum was retained and then sorted to remove duplicates, the number of tuples which are substituted could be reduced. So, instead of having to do the remaining query $|R(S)|$ times, it would only have to be done once for each unique value in the pnum column of SUPPLY.

Only selecting the columns referenced by the query will never reduce the number of tuples, it only reduces the tuple width. This operation is equivalent to sending a one-variable query over the SUPPLY relation to OVQP which has a target list containing only the domain pnum and no qualification. OVQP does not check for duplicates when creating the result relation, it simply copies the appropriate column from the original relation. Thus, an additional operation is required to remove duplicate tuples. If a variable which is to be substituted had a one-variable restriction which was preprocessed, the selection of referenced columns took place at that time so only the operation to remove duplicates would be required prior to substitution.

6.3.1 Analysis of Removal of Duplicates

In the INGRES system there are two available methods for removing duplicates. The first is to hash the relation, checking for duplicates, and the second is to sort. For a complete discussion and comparison of these methods within INGRES, see Appendix B. From the results presented there, it is clear that whenever the tuple width is small, sorting is less expensive than hashing even when the number of tuples is large. Since the operation to remove duplicates is a concomitant of projection, it is likely that the projected tuple width will be small. For this reason, in the general case, sorting will be used to remove duplicates.

However, it is possible to define an upper bound on the width and whenever that bound is reached, switch to hashing as the method for removing duplicates.

First, a two-variable query will be examined to determine under what conditions projection prior to substitution would be advantageous. Then, it will be shown how these two-variable results generalize to an n-variable query.

IPAGE is an INGRES-defined constant for page size; p is the percentage of duplicates in the reduced-width relation (so, $q=1-p$ is the percentage of non-duplicates), $0 \leq p \leq 1$. UPAGE and BUF are as defined in Appendix B. Note that $1 \leq w_i' \leq w_i \leq \text{IPAGE}$. It is assumed that variables are selected for substitution according to their size in tuples.

The cost of processing if X_1 is substituted and no projection is performed is

$$C = t_1 + t_1 P_2$$

The cost of processing if X_1 is substituted and a projection is performed on R_1 is (refer to Appendix B for a description of the sorting cost)

$$C_p = \text{selection cost} + \text{modify cost} + qt_1 + qt_1 P_2$$

$$C_p = (P_1 + P_1') + (2P_1' + 4S_1' + 2S_1'(\text{int}(\log_7(\frac{S_1'}{BUF} - 1) + 1)))$$

$$+ qt_1 + qt_1P_2$$

where S_1 refers to the number of UNIX pages required for relation R_1. (Appendix B explains the difference between UNIX pages and INGRES pages.)

$C_p \leq C$ if and only if

$$q \leq 1 - \frac{1}{1+P_2}\left|\frac{1}{c_1} + \frac{3}{c_1{'}} + \frac{2}{k_1{'}}(2+\mathrm{int}(\log_7(\frac{S_1{'}}{BUF}-1)+1))\right|$$

where k_1 is the tuple capacity of a UNIX page = $\frac{UPAGE}{w_1}$.

The lowest value for the right-hand-side of the inequality is when the quantity being subtracted is as large as possible. This occurs when c_1,$c_1{'}$, and $k_1{'}$ are as small as possible. The smallest value c_1 can be is 1; since sorting is the method being used to remove duplicates, this implies that $w_1{'}$ is small, say less than half of the page size, so the smallest $c_1{'}$ and $k_1{'}$ can be is 2. For these values, the bound becomes

$$q \leq 1 - \frac{1}{1+P_2}(5.5 + \mathrm{int}(\log_7(.01t_1-1))) \text{ for } BUF=50$$

For reasonably large P_2 (P_2>100), the term $(5.5+\mathrm{int}(\log_7(.01t_1-1)))/(1+P_2)$ approaches zero since $\mathrm{int}(\log_7(.01t_1-1)) << P_2$. Therefore, as P_2 gets large, the bound on q approaches 1.

Take the case t_1=10^5. The percentage of duplicates needed for q to be below the bound is

% of dups	P_2
8.4	100
1.7	500
.849	1000
.17	5000

which are reasonable numbers.

When the restricted tuple width is large, so that hashing should be used, the results are

$C_p \leq C$ if and only if

$$q \leq 1 - \frac{1}{1+P_2}\left|\frac{1}{c_1} + \frac{2}{c_1'} + 2.888\right| \leq 1 - \frac{5.888}{1+P_2}$$

This bound is also quite close to one for large P_2.

The case when a restriction has already been performed on R_1 so that only the operation to remove duplicates is required differs from the previous case only in that the first two terms of the cost function associated with the selection cost are no longer included. So,

$C_S \leq C$ if and only if

$$q \leq 1 - \frac{1}{1+P_2}\left|\frac{2}{c_1'}+\frac{4}{k_1'}+\frac{2}{k_1'}(\mathrm{int}(\log_7(\frac{S_1'}{BUF}-1)+1)\right|$$

Using the same reasoning as in the first case, the right-hand-side becomes

$$q \leq 1 - \frac{1}{1+P_2}(4 + \text{int}(\log_7(.01t_1-1)))$$

Since the constant term in the numerator is smaller than the corresponding case including the selection operation, this bound on q approaches even closer to 1 as P_2 gets large.

For the restricted width large and thus the use of hashing to remove duplicates,

$C_S \leq C$ if and only if

$$q \leq 1 - \frac{1}{1+P_2}(\frac{1}{c_1} + 2.888) \leq 1 - \frac{3.888}{1+P_2}$$

which is a bound also quite close to 1 for large P_2.

These analyses of the two-variable case show that the number of duplicate tuples needed to make the projection or sorting worthwhile is quite small. So, the operation of removing duplicates would be beneficial whether statistics were available or not.

To generalize these results to the n-variable case, note first that this two-variable query is the final query in an n-variable query (n>2). That is, a three-variable query after substitution for one of its variables would be a family of two-variable queries. So, that in the inner loops of substitution the analysis above can be applied directly.

At the outer levels of an n-variable query (n>2), the cost function would be of the same form as above with an additional cost reflecting the substitution of the first n-2 variables. For example, the cost of a three-variable query without projection is

$$C = t_1 + t_1(t_2 + t_2P_3)$$

Considering projection prior to substituting for X_1 changes the bound on q only in that it increases the denominator of the term which is substracted from 1, i.e.,

$$q \leq 1 - \frac{1}{1+t_2+t_2P_3}(\text{expr})$$

resulting in the right-hand-side of the inequality becoming even closer to 1.

Thus, the conclusions reached for the two-variable case generalize to the n-variable case in that having statistics available would not greatly effect the decision to project.

There is one case where the projection should not be performed. If it is known that the operation of selecting the referenced columns will cause all columns of the original relation to be retained, the projection should not be performed since, by definition, relations do not contain duplicate tuples.

One other comment should be made. This step should be dissociated from the process of selecting a variable for substitution. Clearly, it is infeasible to do this at each level prior to selecting a variable because of the cost involved. It can be done after the variable has been selected under the premise that a good variable has been chosen and this operation will make it even better. Or, it could be done once at the beginning of processing for the entire query. This would involve projecting all relations involved in the query. It is felt that the first option is better since this allows for combining the retention of columns operation with a possible restriction even though the true size (without duplicates) would not be available to the variable selection process.

6.4 Reformatting

Consider the following three-variable query.

FIND THE EMPLOYEES WHO HAVE MADE SALES AND THE ITEMS THEY SOLD

```
     RANGE OF (E,I,S) IS (EMPLOYEE, ITEM, SALES)
     RETRIEVE (E.name, I.name)
     WHERE E.number=S.employee  AND  S.inumber=I.number
```

If S were substituted for first, this would result in a family of queries of the form

```
     RETRIEVE (E.name, I.name)
     WHERE  E.number=β  AND  β=I.number
```

where $\beta \in$ SALES.

This query has two one-variable restrictive clauses which, according to Section 6.2, should be performed prior to continuing. It is known that these one-variable subqueries are going to be executed ¦R(S)¦ times, that is, once for each tuple in SALES. If the relations EMPLOYEE and ITEM were modified to keyed structures on the appropriate domains, OVQP would be able to process the one-variable subqueries more efficiently. This operation is called reformatting. It is possible to reformat after every substitution which produces any one-variable clauses.

There are certain drawbacks to reformatting however. For example, in the above query, if the EMPLOYEE relation is to be reformatted, a copy must be made of it first since it is the user's relation. Then, the copy of the relation can be keyed on the domain number. Both of these operations have significant costs associated with them. And, if the SALES relation has very few tuples which must be substituted, the cost associated with reformatting could far outweigh the benefit gained. Also, if EMPLOYEE has only a small number of tuples, the cost of reformatting might be too great to warrant its use. Thus the cases where the cost outweighs the benefit must be recognized.

If a user's relation is to be reformatted, it was mentioned that a copy would have to be made first. This copy could be made retaining only the domains referenced by the query since there is no extra cost associated with selecting specific columns. And, by retaining only required columns, the tuple width will most likely be reduced and thus the number of pages required to store the relation will decrease. Since OVQP works with pages, this operation alone could have sufficient effect to warrant the structure modification unnecessary. This reformatting option will be discussed in sub-section 1.

Clearly, if the original relation is already keyed on a useful domain, reformatting should not be performed. The analysis to follow assumes that the original relation has no useful structure for processing the current one-variable subquery.

If it is desired to modify the storage structure of a relation to allow for more efficient retrieval of its tuples, there are two keyed structures available within INGRES. The relation can be hashed on a given key or sorted on a given key (ISAM). Each of these structures are useful under certain conditions. If a tuple is being retrieved searching for a specific value (i.e., equality constraint), then hashing is usually more efficient. If a tuple is being retrieved to determine if it lies within a range of values (i.e., comparative constraint), then ISAM is better. Both choices are available to reformatting and its decision should be based on the characteristics of the restrictive clauses to be executed. The case when hashing is the best structure is presented in sub-section 2 and the use of an ISAM structure is analyzed in sub-section 3.

IPAGE is an INGRES-defined constant for the page size; $1 \leq w_i' \leq w_i \leq$ IPAGE. When the relation has been modified to a keyed structure, the tuples with the same key values will be clustered and thus the number of pages which must be accessed to find a specific key value can be limited to less than the total number of pages. $p_i = E[p_i(\beta)]$ is the expected percentage of pages of R_i which will be accessed to verify the constraint in X_i and $\beta \in R_j$, where X_j is the variable previously substituted, assuming that R_i has been modified to an appropriate keyed structure. For a discussion of the costs used for the operations of modifying a relation to a hashed or ISAM structure, refer to Appendix B.

6.4.1 Selection of Referenced Domains

The operation of retaining only the referenced domains while making a copy of the user's relation has essentially no overhead as compared with simply making a copy. Since OVQP works on units of pages, if more tuples can be fit on a page by reducing the tuple width, a large savings can be gained by doing this selection operation even without a subsequent modify operation. Clearly since this operation does not eliminate any tuples, the total number of tuples in the relation will remain unchanged. And, since a convenient access structure has not been imposed on the reduced width relation, OVQP will have to examine every tuple to verify the constraint. Thus, for only selecting the needed domains, statistics on the percentage of tuples satisfying a given restriction will not play a role in determining the benefit of this operation. The important factor here is the reduction in tuple width and thus a reduction in total pages for the relation. So, this discussion will not affect the overall conclusion as to the benefits gained from having the statistics available, but it is included for completeness of the analysis of reformatting.

A two-variable query in X_1 and X_2 will be examined since the costs will appear in the same form within an n-variable query. The cost of processing using only tuple substitution and assuming that X_1 is substituted is

$$C = t_1 + t_1P_2$$

The cost of processing if the appropriate domains of R_2 are selected is

$$C_p = t_1 + \text{selection cost} + t_1P_2'$$

$$C_s = t_1 + P_2 + P_2' + t_1P_2'$$

$C_s \leq C$ if and only if

$$\frac{P_2'}{P_2} \leq \frac{t_1-1}{t_1+1}$$

which is equivalent to $\frac{c_2}{c_2'} \leq \frac{t_1-1}{t_1+1}$ since $P_2=\frac{t_2}{c_2}$ and $P_2'=\frac{t_2}{c_2'}$. Clearly, $\frac{c_2}{c_2'} \leq 1$ since $w_2' \leq w_2$. Also, if $w_2'=w_2$ which implies $c_2'=c_2$, this operation should not be performed since $\frac{t_1-1}{t_1+1} < 1$.

As w_2 increases, more reduction in width is required so that the reduction will have an effect on the number of pages. For example, when $w_2 > \frac{\text{IPAGE}}{2}$ only one tuple will fit on a page, so for the selection of columns operation to be beneficial $w_2' < \frac{\text{IPAGE}}{2}$.

The conclusion of the analysis is that, since all quantities are known at the time the decision is to be made, the calculations should be performed to determine if $\frac{c_2}{c_2'} < \frac{t_1-1}{t_1+1}$. If so, reduce the tuple width; otherwise eliminate this operation alone as an alternative. A decision to perform the selection of columns does not eliminate the possiblity of doing a subsequent modify. All it says is, that if the modify is determined to be not worthwhile, the operation to reduce the tuple width should still be performed. Likewise for the decision to not perform the selection of columns - the decision does not preclude doing this operation in connection with a subsequent modify.

6.4.2 Reformatting to a Hashed Structure

The following analysis will use the assumption that a variable is selected for substitution on the basis of its size in tuples. Thus, for the order of substitution $X_1, X_2, \cdots, X_n$, $t_1 \leq t_2 \leq \cdots \leq t_n$. In the cost functions presented here, the cost of modifying a relation to a hashed structure will use a value for an "average" page for the overhead associated with overflow pages (0.888, refer to Appendix B). In practice, this additional cost can be computed using the tuple capacity of a page in the relation under consideration.

Consider first a two-variable query in X_1 and X_2 with $t_1 \leq t_2$. The cost of processing without reformatting is

$$C = t_1 + t_1 P_2$$

The cost if reformat, including selection of the referenced columns while making a copy, is performed is

$$C_R = t_1 + \text{copy cost} + \text{modify cost} + t_1 p_2 P_2'$$

$$= t_1 + (P_2 + P_2') + (P_2' + 2.888 t_2) + t_1 p_2 P_2'$$

$C_R \leq C$ if and only if

$$p_2 \leq \frac{c_2'}{c_2} - \frac{1}{t_1}\left|\frac{c_2'}{c_2} + 2.888 c_2' + 2\right|$$

Since $w_2' \leq w_2$, $c_2' \geq c_2$ so that $\frac{c_2'}{c_2} \geq 1$. Thus, the right-hand-side of the inequality is smallest when $w_2 = w_2' = 1$ and the dominant term in C_R becomes $2.888t_2$. This results in

$$p_2 \leq 1 - \frac{335.888}{t_1}$$

These results can be expanded to the n-variable case in the form that to reformat the ith relation (including the copy),

$$p_i \leq \frac{c_i'}{c_i} - \frac{1}{t_{i-1}} \left| \frac{c_i'}{c_i} + 2 + 2.888c_i' \right| \leq 1 - \frac{335.888}{t_{i-1}} = \bar{p}_i$$

where t_{i-1} refers to the variable just previously substituted, that is, the order of substitution is $X_1, X_2, \cdots, X_n$.

Since $t_{i-1} \leq t_i$,

$$\bar{p}_i = 1 - \frac{335.888}{t_{i-1}} \leq \bar{p}_{i+1} = 1 - \frac{335.888}{t_i}$$

Therefore, $p_i \leq \bar{p}_i$ is satisfied if $p_i \leq \bar{p}_2 = 1 - \frac{335.888}{t_1}$, $i=2,...,n$.

However, if $t_1 < 336$, this results in a negative lower bound on p_i which is clearly infeasible. This illustrates that there is a threshold value on t_1 in terms of c_2 and c_2', and, if t_1 is below this threshold, reformatting should not be considered.

This equation in p_2 can be used to define a threshold value on t_1 given an upper bound A on p_2. That is, by solving the right-hand portion of the following inequality for t_1

$$p_2 \le \frac{c_2'}{c_2} - \frac{1}{t_1}\left|\frac{c_2'}{c_2} + 2.888c_2' + 2\right| < A$$

a threshold value for t_1 can be obtained. Research done by Lum et al [LUM71] and Severance and Duhne [SEVE74] indicates that on the average there will be 1.5 page accesses for a single tuple retrieval (using hash with chained overflow). Thus, $A=.10$ is a reasonably high bound for more than 15 pages. This results in a condition on t_1 of

$$t_1 \ge \frac{c_2}{c_2'-.1c_2}\left|\frac{c_2'}{c_2} + 2.888c_2' + 2\right|$$

which is an easily calculatable bound. The largest value for this bound occurs when $w_2=w_2'=1$ and results in $t_1 \ge 372$.

The conclusion is that setting a threshold on t_1 would be of more use to reformatting than trying to use statistics to estimate the percentage of pages which must be accessed for a given key value. This threshold could be calculated for each instance of reformatting since w_2, w_2' and page size are known quantities. This threshold formula could be calculated assuming $A=.20$ which is reasonable whenever the number of pages in R_i is more than 7. A similar bound on t_1 can be calculated when it is not necessary to make a copy of the user's relation.

Currently the INGRES system supports the option of reformatting to a hashed structure. The one major drawback discovered in practice has been in the case when one of the relations involved is fairly small. It was found that by inserting a crude cost function which used a fixed value of p_i it was possible to filter out those cases when the ranges involved were too small to warrant reformatting. This experience supports the conclusions reached by the analysis.

6.4.3 Reformatting to an ISAM Structure

This analysis also makes use of the assumption that a variable is selected for substitution on the basis of its size in tuples. Thus, for the order $X_1, X_2, \ldots, X_n$, $t_1 \leq t_2 \leq \cdots \leq t_n$.

Consider first the two-variable case. The original cost, without reformatting, is

$$C = t_1 + t_1 P_2$$

The cost if reformatting and a copy are performed is

$$C_R = t_1 + (P_2 + P_2')$$

$$+ (2.25P_2' + 4S_2' + 2S_2'(\mathrm{int}(\log_7(\frac{S_2'}{BUF} - 1) + 1)))$$

$$+ t_1 p_2 P_2'$$

where S_2 refers to the number of UNIX pages for R_2.

$C_R \leq C$ if and only if

$$p_2 \leq \frac{c_2'}{c_2} - \frac{c_2'}{t_1}\left|\frac{1}{c_2}+\frac{3.25}{c_2'}+\frac{4}{k_2'}+\frac{2}{k_2'}\text{int}(\log_7(\frac{S_2'}{BUF}-1)+1)\right| \qquad (A)$$

where k_2 is the capacity of a UNIX page.

Now consider that c_2 is the original capacity of a page in R_2 and c_2' is the new capacity after the width has been reduced. So, $\frac{c_2'}{c_2}$ is the gain in tuples per page achieved by the domain selection operation. Define $g = \frac{c_2'}{c_2}$. Equation (A) can now be rewritten as

$$p_2 \leq g - \frac{1}{t_1}\left|g+3.25+\frac{4c_2'}{k_2'}+\frac{2c_2'}{k_2'}\text{int}(\log_7(\frac{S_2'}{BUF}-1)+1)\right|$$

$$\leq g - \frac{1}{t_1}\left|g+3.25+4+2\text{int}(\log_7(.01t_2-1)+1)\right|$$

Assume that $t_2=10^6$ (as t_2 decreases, $\log_7(.01t_2)$ also decreases), then

$$p_2 \leq g - \frac{1}{t_1}(g + 17.25)$$

If $g=1.2$, or there is a 20% increase in tuples per page after the reduction in tuple width, then

$$p_2 \leq 1.2 - \frac{18.45}{t_1} \qquad (B)$$

If $t_1 \geq 100$, (B) becomes $p_2 \leq 1$ which is always true. Clearly, the lowest bound occurs when $g=1$ so that $p_2 \leq 1 - \frac{18.25}{t_1}$. This formulation for the bound of p_2 illustrates that if the width reduction had any effect on the number of tuples per page, the modification to an ISAM structure will be beneficial for large t_1 regardless of the statistics. Even in the case where there is no reduction in width, for large t_1, the bound is very close to 1.

In the case where the copy operation is unnecessary,

$C_M \leq C$ if and only if

$$p_2 \leq 1 - \frac{c_2'}{t_1}\left|\frac{2.25}{c_2'} + \frac{4}{k_2'} + \frac{2}{k_2'}\mathrm{int}(\log_7(\frac{S_2'}{BUF}-1)+1)\right|$$

$$\leq 1 - \frac{1}{t_1}\left|2.25 + 4 + 2\mathrm{int}(\log_7(.01t_2-1)+1)\right|$$

When $t_2=10^6$, $p_2 \leq 1 - \frac{16.25}{t_1}$.

This analyses can be generalized to the n-variable case in the same manner that was used in the section on hashing.

It might seem that the effectiveness of modifying to an ISAM structure depends highly on the reduction in width gained during the copy, which would imply that selection of domains is the important operation and little more benefit is gained by the modify. But compare the cost of doing only a copy with retention of referenced domains to the cost of doing a copy and modify. This is equivalent to performing only a modify assuming the copy has already been done. The analysis of this case shows that the modify has a large benefit of its own.

The conclusion is that whenever $g \geq 1.1$ reformatting to an ISAM structure should be performed (for $x_1 \geq 185$ tuples). Even when $g = 1$, the bound on p_2 is high enough so that whenever x_1 is reasonably large ($x_1 \geq 100$ tuples) reformatting to an ISAM structure would be beneficial.

6.5 Creating Secondary Indices Dynamically

A secondary index (inversion) on a domain is a binary relation between values of the domain and tuples (or tuple identifiers) from the data relation. Given a relation R, the following query will create an index on domain a of R:

```
RANGE OF X IS R
RETRIEVE INTO AINDEX(X.a, ptr=X.tid)
```

Clearly, a secondary index will have the same cardinality as the relation it is indexing, but usually it will occupy less pages since the tuple width will be smaller.

If an index is used in the processing, then, when a value is supplied, the index is accessed. By only accessing the index, it is possible to obtain the tuple identifiers of all tuples in the original relation which have that key value. In essence, the existence of an index transfers the search for satisfying tuples to the index rather than the original relation.

A combined index, as described in [LUM70] and [MULL71], is an approach which takes advantage of several domains being specified. Here the index relation consists of a projection of several domains and the tid from the data relation instead of just a single domain as in a simple index. Since this discussion concerns dynamic creation of indices and it will be known at creation time which domains will have values specified, this type of index can be useful whenever the condition to be verified does not contain disjunctions.

The other option available when several domains are involved in the constraint is to use list processing techniques to combine the qualifying tids obtained from indices on each domain individually. This technique is not currently supported by INGRES.

Note that when a condition such as "X.a > $value_1$ OR X.b < $value_2$" is to be verified, the only technique which will limit the tuples to be scanned is a simple index on each of the domains involved. The qualifying tids from each index can then be unioned to produce the set of all qualifying tids. A combined index on both domains will be of no help and neither will modifying the structure of the primary relation so that it is keyed on the concatenation of both domains. Since list processing techniques to combine two or more indices are not supported by INGRES, conditions such as the above containing disjunctions will be excluded from the following discussion.

Consider the situation where it is known that a one-variable restriction will be executed many times due to a previous substitution. It would be advantageous to make the evaluation of this one variable query as efficient as possible. One available technique, which has already been discussed, is to dynamically reformat the relation involved to a keyed structure on the referenced domains. The purpose of this section is to present an alternative to reformatting, namely dynamically creating an index on the referenced domains, and to compare the effects of these techniques.

Examine the one-variable query over a relation R which is to be executed. It will be of the form

RETRIEVE T(R)

WHERE domain_1=value_1 AND domain_2=value_2 AND ... AND domain_k=value_k

(the "=" can also be an inequality) where T(R) is a function of the domains of R which are to be retrieved.

Consider the costs associated with evaluating this only once. If R has no primary or secondary access paths which can be used, it will be necessary to examine every page of R to determine the qualifying tuples. If the structure is such that R is keyed on $\text{domain}_1,\cdots,\text{domain}_k$, then the qualifying tuples can be accessed directly. Thus only a single page will have to be retrieved if there are only a few tuples with the specified values. Using a secondary index on $domain_1, \cdots, domain_k$, a page of the index will be accessed and then the page of the primary relation containing the qualifying tuple will have to be retrieved.

Suppose this one-variable query will be executed n times. With no structure, nP(R) pages will be accessed, where P(R)=no. of pages in R. If exactly one tuple satisfies for each set of values, using a keyed structure 1n pages must be retrieved while 2n pages are required using an index. It can be seen clearly that the cost of accessing the tuples is less using a keyed structure for the primary relation.

However consider now the cost of providing the two access paths. To modify the relation R to a structure keyed on $domain_1, \cdots, domain_k$ requires retaining $domain_1, \cdots, domain_k$ and any other domains referenced by T(R). Then this resulting relation must be sorted or hashed on $domain_1, \cdots, domain_k$. If it is to be hashed, the cost is on the order of 2|R| page accesses while if it is to be sorted, the cost is on the order of P(R)logP(R) page operations. To create the index, only $domain_1, \cdots, domain_k$ must be retained along with the tuple identifier and then sorted. If T(R) references several more domains than $domain_1, \cdots, domain_k$, the number of pages in the index will be quite smaller than P(R), and thus the sort, which is on the order of P(I)logP(I) operations, will be less expensive.

Combining the cost of providing the access path and accessing the tuples in the case where exactly one tuple satisfies for each set of values, unless n is much larger than P(R), the secondary index will be the access method preferred. This will also be true when there are certain sets of values for which no tuples satisfy or when only a few tuples satisfy, for example the number of tuples which fit on a single page of R.

The next case to consider is when many tuples qualify for each set of values for $\text{domain}_1, \cdots, \text{domain}_k$. This can happen if some of the clauses are inequalities or if one of the domains is such that many tuples have the same value. The costs of providing the two access paths will remain the same but the cost of accessing the qualifying tuples will change. Using a keyed structure, the data pages will be "clustered" with respect to the values of $\text{domain}_1, \cdots, \text{domain}_k$. That is, in accessing the pages containing qualifying tuples, each page will only be accessed once and a minimum number of pages, usually less than P(R), will have to be retrieved. However, using an index, first an access is made to the index resulting in the tid of a tuple that qualifies, and then this tid is used to access the relation R. When many tuples qualify, this results in random references to the pages of the data relation R. Usually this will require a page fetch for R for each tuple that qualifies.

To illustrate this difference, consider a single execution of the one-variable subquery where m tuples satisfy. Using the keyed structure, this will require approximately $\frac{m}{c}$ page accesses to R, where c is the tuple capacity of a single page. With an index, m page accesses to R will usually be required, plus the additional accesses to the index. If c is taken to be only about 20, it can be seen the considerable effect that the clustering property has on performance. When this effect is multiplied by n, the difference can be very large.

For these reasons, when multiple tuples are to be retrieved for most sets of key values, the advantage of the clustering effect gained by reformatting the data relation will usually outweigh the extra cost incurred to achieve it.

These observations illustrate though that the distribution of the key values would be valuable in determining whether to create an index or to reformat the data relation. The decision depends upon each query and the key domains involved.

6.6 <u>Comparison of Available Statistics</u>

If it is decided that statistics should be available for decision making there is a wide range of choices as to exactly what information is used and how it is made available. These choices fall into two categories. The first is information which is gathered when each relation is created and is then updated whenever the relation is updated, and the second is information obtained by sampling the appropriate subset of the database dynamically.

Within the first category there are many choices as to the exact information which is gathered and maintained. Basically, there will be a trade-off between the cost of updating the statistics and their accuracy in depicting the behavior of the data. The statistics within this first category will assume that each domain is independent of all other domains even though this may not always be a reasonable assumption. But, if this assumption is not made, joint distribution statistics would also have to be maintained for all conceivable combinations of domains. Clearly, this would be an unreasonable amount of information to maintain.

The simplest statistics to keep in terms of maintenance costs are probably the maximum and minimum value for each domain of each relation. When the relation is created and the data entered, this would require a maximum of two comparisons per domain per tuple. If the relation is updated, to update the maximums and minimums would again require a maximum of two comparisons per updated tuple for every domain affected by the update. It would be possible to add this information to the system ATTRIBUTE relation which already contains certain characteristics of each domain. If character domains were encoded in some manner, it should be possible to store this information using two words for each value.

But now consider using the maximum and minimum as statistics. If it is desired to know the percentage of tuples from a relation R which have domain a equal to some value β, the distribution of values between the maximum and minimum must be assumed. If an even distribution is assumed, then this percentage is simply the number of tuples divided by the number of possible values. First of all, this is undefined if the domain under consideration is a floating point domain (i.e., a real number) since the number of values is infinite. Second, if the actual data does not fit the assumed distribution, the information gained is more than likely quite inaccurate. The conclusion is that even though a maximum and minimum are very inexpensive to maintain, the statistics provided by them alone are not accurate enough for optimal decision making. But this points out the fact that some basic knowledge of the distribution of values for each domain would be helpful.

Suppose that a count of the number of different values occurring within each domain was maintained. The storage required by this information would be minimal (namely, two words per domain) and this could also be stored with other characteristics of the domain in the system ATTRIBUTE relation.

However the updating costs would be quite high since every time an update is performed, the entire relation would have to be read to recount the number of different values. This is because the values themselves are not being kept, so when a tuple is added, deleted or changed, it is not known if the values in that tuple are unique to that tuple or not.

However, this information alone is not much more useful than the maximum/minimum values, since it still must be assumed what the distribution of tuples among the possible values is. So, if it is known that there are 10 different values for domain a and that there are 100 tuples, first it must be assumed that β is one of those 10 values. This in itself may not be true. Then, it must be assumed that, for example, these 100 tuples are evenly distributed among the 10 values. Again, if the actual data does not fit the assumed distribution, the indication of behavior used by the decision-making process may be quite inaccurate.

Even if the maximum and minimum values together with a count of different values were maintained, an assumption would still have to be made as to the actual distribution of the tuples. So nothing much is gained by keeping both sets of information.

The next logical step in information is keeping a type of histogram associated with each domain. This would make available the number of different values and the number of occurrences of each value. Actually, this information is easier to update than the count of different values because it is not necessary to scan the whole relation to determine the effect of an update. It is only necessary to adjust the count of occurrences for the value appearing in each domain of the updated tuple. However, there could be difficulties when a new value is added or a value no longer occurs. This depends on the type of histogram which is maintained. Basically, there are two choices - a fixed interval histogram or a variable-interval histogram.

For a fixed interval histogram, it is defined a priori that there will be n equal-sized intervals and the values falling within each interval are determined by the maximum and minimum. For an example see Figure 6.1. This type of histogram is easy to update unless it is the maximum or minimum which is changed. This would require that the whole histogram be recalculated if the limits for each interval are affected. This type of histogram also requires a fixed amount of storage, namely a maximum, minimum for the whole domain and a count for each of the n intervals since the

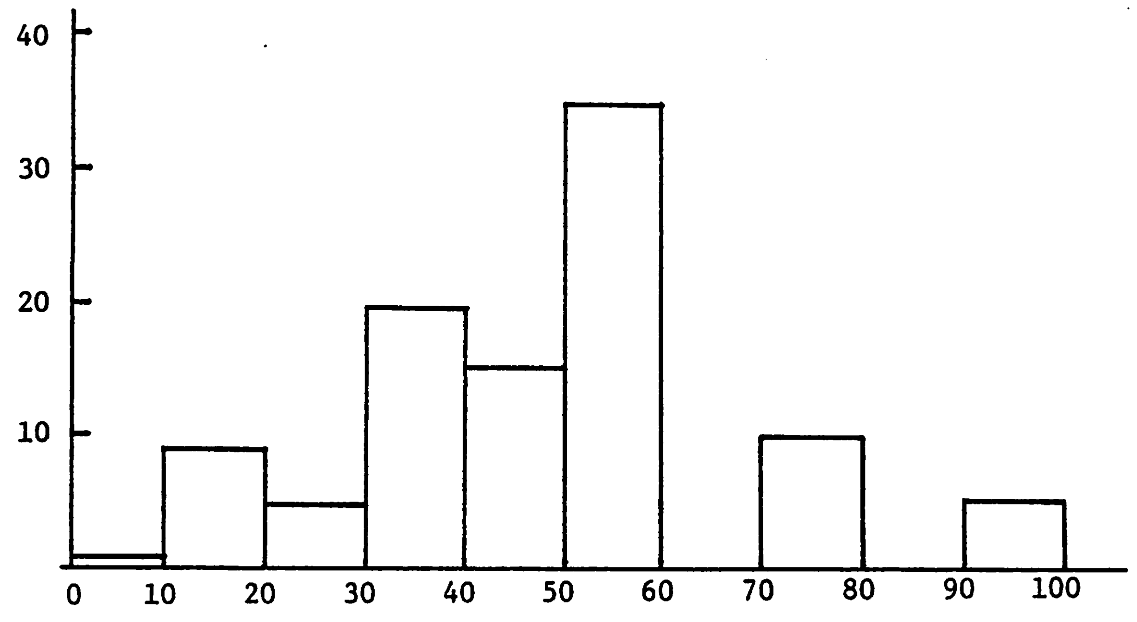


Figure 6.1. Fixed-interval histogram.

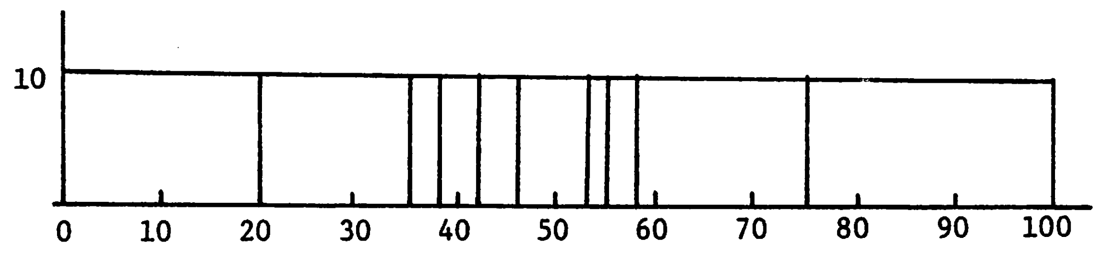


Figure 6.2. Variable-interval histogram.

limits on each interval can be determined from the maximum and minimum values. However, if the range between the maximum and minimum is quite large, then certain information can get lost, particularly if n is small. It may happen that certain intervals have a very large number of occurrences while others have none. And, if the range of a given interval is large, then there is still the problem of assuming a distribution for the number of occurrences within the interval to the available values. True, by breaking the entire range into intervals, the error should not be as great as that using only a maximum, minimum and count for the whole domain, but it is still possible there will be a large discrepancy between the estimated value and the actual value. For this type of histogram, it is unreasonable to make the number of intervals equivalent to the number of possible values unless this number is quite small, mainly because of the storage involved.

The second type of histogram proposed will be called a variable-interval histogram. Basically, the limits for each interval are variable but the number of occurrences within each interval is fixed. Define n_0 as the number of occurrences for each interval; then a sequence k_i, $i=1,\ldots,m$ defines the interval limits such that $n(k_i,k_{i+1}) = n_0$. Figure 6.2 illustrates this type of histogram for the same domain described in Figure 6.1. The main advantage of this technique is that in areas where the values are very dense, the interval limits will be smaller so this histogram depicts the true distribution more accurately. The number n_0 can be determined by some function of the total number of tuples and this determines the number of intervals.

This histogram requires only slightly more storage than the fixed interval histogram. Namely, the value n_0 must be stored, the maximum and minimum values must be stored and a limit for each interval. Also, this type of histogram is more difficult to update. When tuples are added or deleted, this can affect the number of occurrences within one interval, namely for some j, $n(k_j,k_{j+1}) \neq n_0$, so that the limits for every interval and possibly the n_0 value must be recalculated. There is an option to updating and possibly recalculating the histogram every time an update is performed to the domain. If a date is associated with the histogram, then the system, during "free time", can compare this date with the date when the relation was last updated and if there is a large discrepancy, recalculate the histogram. In this way, no extra cost or delay is added to the user's updates, but the penalty is that the statistics available are possibly inaccurate at times.

It should be mentioned that both types of histograms are best used for determining tuples which fall within a specified range of values. To answer the question of the number of tuples equal to a specific value requires more exact information than a histogram supplies, unless of course each interval contains only a single value. But to answer this question of equality really requires a list of all values and the number of tuples containing that value, which is infeasible in the general case. And, this would be tantamount to maintaining an index for every domain.

If the cost of storing and maintaining the statistics is to be avoided, then there must be a method of obtaining the statistics dynamically when they are needed. This would require either calculating the statistics discussed previously "on-the-fly" or sampling the subset of the database involved in the current query. Since calculating the statistics on-the-fly would be equivalent to answering the query, this does not seem like a logical choice.

Sampling the database is equivalent to answering the query on a small subset of the data. If the amount of data involved in the original query is quite large and sampling determined the optimal processing strategy, the overhead involved in the sampling might be quite insignificant compared to the cost of processing by a sub-optimal path. However, if the amount of work involved in answering the optimal query is not significant, clearly sampling should not be used. There should be some type of threshold, defined perhaps by the size of the cross product involved or an estimate of total worst case processing time, which would limit the cases where sampling could be considered.

The major problem associated with sampling is defining what the sample should be. First of all, the size of the sample must be determined considering storage cost vs. accuracy. Second, the tuples which are included in the sample should be "random" tuples. So, if the relation is sorted, it is not sufficient to select the first n tuples to be included in the sample. This implies that the organization of the original relation (i.e. storage structure and keys) must be considered. Obviously, if the sample selected is not a good one, the information obtained from it could be misleading in selecting a processing strategy. The cost of sampling involves creating and filling a sample relation for each relation involved in the current query, and then, processing the query for those sample relations. It should be mentioned that in order to take full advantage of the sample and make an "optimal" decision, it is possible to process more than one query at each decision point. That is, the same query may be run more than once in order to compare options. For example, rather than deciding that variable X is going to be substituted and then sampling to determine if projecting and removing duplicates would be worthwhile, it is possible to project and remove duplicates on all the sample relations and then determine which one should be substituted (i.e., which one has the fewest tuples). If the sample is going to be made anyway, full advantage should be taken of the information contained in it.

Conclusions. If the exact distribution of all data is known by the user when the relation is created and this distribution will not change as the data is updated, then the user could inform the system of this information. However, relying on the user to know the distribution of all of his data is not usually feasible. If it is determined that statistics should be available to the system, then it seems that either a variable-interval histogram should be maintained or sampling should be used if the size of the database warrants it. This decision is based mainly on the fact that the accuracy of the information made available by these methods is much more than that of only slightly less expensive methods.

CHAPTER 7

VARIABLE SELECTION

In the general decomposition algorithm presented in Chapter 2, a variable is chosen for substitution on the basis of the number of tuples in its range relation. It has yet to be shown if this is a good or even reasonable selection criterion. The selection of a variable for substitution is a very critical step in the processing of a query. If a wrong choice is made, it can have catastrophic results on the total processing cost. The purpose of this chapter is to discuss the parameters involved in the cost of processing a query and how these parameters can be used to develop a good selection criterion.

Within most other relational systems, an execution plan for processing the entire query is determined before any actual processing begins. System R [ASTR76] does this so they can compile queries and thus avoid the optimization overhead in a production environment. The policy opted for in the INGRES system is to make the decision process a dynamic one. Thus, at each phase of the processing a decision is made only as to what the next step should be. In this way, the current decision takes into account the effect of the last decision on the query environment. It is felt that this allows more flexibility and thus more possible optimization in the processing.

Using the dynamic decision policy, there are several factors which will influence the decision made at each step. These include the size of the relations involved, both number of tuples and number of pages, the storage characteristics of the relations which determine the available access paths, the structural characteristics of the query itself, and any distributional information which is known about the domains referenced in the query. All of these factors except the last depend upon information which is readily available and thus do not incur much cost to be included in the decision process. The distributional information could involve a large cost for storage and maintenance but it also could provide a much more accurate prediction for the effect of a particular decision.

The following sections will consider the effect of these factors on the choice of a variable for substitution. Results of experiments run on queries using various selection criterion are presented in Chapter 8. In the first section, a discussion of two-variable queries will be presented. This is included as a special case for two reasons. First, the initial decision in a two-variable query essentially determines the execution plan for the entire query thus allowing for a prediction of the entire cost. Second, in the processing of any multivariable query, at some point a two-variable query will be involved. So, by executing two-variable queries in an efficient manner there is a better chance for processing three or more variable queries with minimal cost.

When a query involves three or more variables, the problem becomes more complex. n variables are available for substitution and reduction becomes an available tactic. The general problem of three or more variable queries and what this step is attempting to do are discussed in Section 7.2. Then, in the next sections, each of the factors will be considered individually and its effect examined on queries involving three or more variables.

The discussion to follow will assume that the decision as to whether one-variable clauses should be preprocessed is independent of the variable selection decision. Thus, if one-variable restrictions are to be preprocessed, they will have been done so prior to this step. But the existence of one-variable clauses is not excluded as a possibility in the proposed estimation procedures. Let Q be a query in variables $X_1, X_2, \cdots, X_n$ with ranges $R_1, R_2, \cdots, R_n$. The following notation will be used throughout the discussion.

t_i	the number of tuples in relation R_i.
c_i	tuple capacity of a page in R_i - the number of tuples in R_i which can fit on a single page of secondary storage (the integer portion of page

size divided by tuple size since INGRES does not allow a tuple to be split between pages).

P_i the number of pages required to store relation R_i.

7.1 Two-Variable Queries

There are several points to recognize concerning two-variable queries before examining their processing. First, the one-variable query was chosen as the basic unit of processing for the INGRES system. Thus the methods proposed in the literature [GOTL75, BLAS75] for computing joins are not directly applicable. They do, however, provide indications as to what structures are most useful. Second, reduction is not an available tactic so tuple substitution is the only alternative to reduce the number of variables. Third, it is known that the relation whose tuples will not be substituted will be accessed by OVQP. Because of this, the unit of its access will be pages and a keyed structure can be used advantageously.

Consider a two-variable query in variables X_1, X_2 with ranges R_1 and R_2. The criteria to select a variable for substitution will be considered in the order of simple to more complex and then the applicability of each criterion will be examined.

(1) Size Only. In this case, to make the decision, it is assumed that there are no useful access paths for either

relation. Using the cost function presented in Chapter 6, which from experiments performed is exact for this situation,

Cost of substituting for X_1

$= t_1 + t_1P_2$

Cost of substituting for X_2

$= t_2 + t_2P_1$

Thus, the variable X_i should be selected which minimizes the ratio $\frac{t_i}{1+P_i}$. This result is similar to Pecherer's results [PECH76] for nested iteration. Both t_i and P_i are known at the time the decision is to be made so it is a simple matter to calculate the ratio for each i and then select the minimum.

Some simple observations can be noted. If $t_1 \simeq t_2$ then the relation with the largest number of pages will be selected. If $P_1 \simeq P_2$, the relation with the fewest tuples is selected. In general, the relation with the smallest tuple capacity per page, or equivalently, the largest tuple width, will be selected for substitution. These results are not surprising.

(2) Size and Query Structure. The structural characteristics of the query which have an effect on a two-variable query are limited and involve only a minor modification to criterion (1). First, by simple examination, it can be determined if the query is disjoint. That is, for

example, the qualification involves only X_1 and the target list only X_2. Obviously in this case, the query should be split into two separate one-variable queries with the one involving the qualification being executed first. The modified decomposition algorithm proposed in Chapter 5 includes recognition of this case as a separate step prior to variable selection.

Execution of one-variable clauses appearing in the qualification is considered as a separate decision from variable selection and thus will not be included here.

The only other point which can be included is examining the qualification to see if it contains any clauses which involve disjunctions. If so, since combination of simple indices is not supported, no access path will be of any help in limiting the scan. Therefore the variable selection can be made on size alone since that will be the determining factor.

(3) Size, Query Structure and Available Access Paths. To examine fully the effect storage structure has on the variable selection process it is necessary to also have statistics available. This case is considered in the next criterion. But, if certain assumptions are made, some general rules can be concluded.

First, assume that $\frac{t_1}{1+P_1} \simeq \frac{t_2}{1+P_2}$ so there is no strong preference as to which variable should be substituted considering only size. Obviously if one of the relations has a keyed access path which can be used, then the other relation should be selected for tuple substitution. This allows OVQP to take advantage of the access path and limit the number of pages which must be fetched.

If both relations have a useful access path, under this assumption the only clear preference is for a keyed structure on the primary relation rather than access via a secondary index. Also, in the case of ISAM structures, if one has an exact key (i.e. all key domains specified) as compared to only a (leftmost) subset of the keys being specified, the first access path will limit the scan the most.

If the distribution of the linking domains is assumed to be uniform, a cost estimating procedure can be used to compare the effect of substituting for each variable. If a query is to be executed once for each of m key values and there are n tuples in the relation to be examined, then $\frac{n}{m}$ tuples will satisfy each execution using the uniformity assumption. Thus the following estimates can be used

Cost of substituting for X_1

$$= t_1 + t_1\left(\frac{t_2}{t_1} \frac{1}{c_2}\right) = t_1 + P_2$$

Cost of substituting for X_2

$$= t_2 + t_2(\frac{t_1}{t_2}\frac{1}{c_1}) = t_2 + P_1$$

This formulation also implicitly assumes that each value substituted in the key domains is unique. Using these estimates implies that the quantity $t_i - P_i$ should be minimized when both relations have a keyed structure. Note that the assumptions used to obtain these estimates are in general not assumptions that fit the true data very well.

(4) Criterion Using Statistics. Here all the previous characteristics will be used and it will also be assumed that complete distributional information is available for the domains involved. If only limited statistics are available, assumptions can be imposed to estimate the required parameters. The interesting case to consider is when one or both of the relations has a useful storage structure. If neither relation is keyed on the linking domain(s), a full scan will have to be made for either choice of substitution variable and statistics will not be of any help.

Assume that X_i is selected for substitution. Let q_j be the expected number of tuples that will satisfy the one-variable query in X_j for each tuple value from R_i. If the access path is direct to the data relation, i.e., an index is not used, then the tuples will be clustered with respect to the values of the key domains. So, approximately $\frac{q_j}{c_j}$ pages of R_j must be accessed to retrieve the q_j qualifying tuples. Thus,

Cost of substituting for X_1

$$= t_1 + t_1 \frac{q_2}{c_2}$$

Cost of substituting for X_2

$$= t_2 + t_2 \frac{q_1}{c_1}$$

If one of the relations has no useful access path, q_j can be set to t_j and the comparisons can still be carried out.

The quantities to be compared are $t_1/1+\frac{q_1}{c_1}$ and $t_2/1+\frac{q_2}{c_2}$ and the variable which minimizes this ratio will be selected for substitution. Clearly, if one of the t_i's is very large compared to the other quantities, this will be the dominant factor and the other variable will be selected for substitution. If one of the t_i's is very small, then this will imply that the associated variable should be substituted. But these are the intuitively obvious conclusions.

Consider the case when one of the variables, say X_1, would be selected if only size were considered. Thus, $\frac{t_1}{1+P_1} < \frac{t_2}{1+P_2}$. But, R_1 also has an efficient access path whereas R_2 doesn't, so $q_2=t_2$. Compare the quantities $t_1/1+\frac{q_1}{c_1}$ and $\frac{t_2}{1+P_2}$, which is asking the question - under what conditions is the access path more dominant than size?

If $\frac{t_2}{1+P_2} < t_1/1+\frac{q_1}{c_1}$ then X_2 will be selected for substitution and the access path for R_1 will be the important factor. But this relationship is true if and only if

$$q_1 < c_1(t_1\frac{(1+P_2)}{t_2} - 1) \qquad (7.1)$$

From this equation it can be seen that if $t_2 \geq (1+P_2)t_1$ then the right-hand-side becomes non-positive, thus imposing an impossible condition on q_1. So size will be the dominant factor.

If $t_2 < \frac{1}{2}(1+P_2)t_1$ then whenever $q_1<c_1$, or at most a single page of R_1 will be accessed for each substitution value, clearly equation 7.1 will be satisfied and X_2 should be selected for substitution. As t_2 gets smaller and approaches t_1, then the bound on q_1 gets larger and approaches P_2c_1.

In the interval $(1+P_2)t_1 > t_2 \geq \frac{1}{2}(1+P_2)t_1$, the implication is that $0 < \frac{t_1(1+P_2)}{t_2} \leq 1$ and since this factor multiplies c_1, c_1 becomes the critical parameter. For small c_1, this could imply the condition $q_1<1$ which is obviously impossible.

This argument has illustrated the use of statistics combined with a knowledge of size and access paths to determine which variable should be tuple substituted in a two-variable query. Note that criterions (3) and (4) can also consider the possibility of reformatting one of the relations in order to provide a more efficient processing plan in making their selection.

7.2 n-Variable Queries (n > 2)

Up to this point, the purpose of this step in the algorithm has been to select a variable which will subsequently be substituted for tuple-by-tuple. However, when considering a query which involves three or more variables, there is also a possiblity of reducing the query into components. But, whether reduction should be used depends upon the role of the variable selected by this step. So, the selection of a variable "for substitution" in actuality determines which processing option, reduction or substitution, will be used.

For these reasons, this decision should be made considering the effects of both options if an attempt is to be made to minimize the processing costs. And, the result of this step will really be either (1) a variable which should be substituted, or (2) the components into which the query should be reduced.

When processing three or more variable queries there are two major reasons why this decision process becomes more complex. First, reduction is an available tactic thus increasing the number of possible processing paths which must be considered. Second, the query involves more variables which means the query itself is generally more complex. For an n-variable query, considering only tuple substitution, there are n! possible orderings for substitution. So, the complexity of the problem can explode quite easily.

7.3 Size as a Selection Criterion

Using only the sizes of the relations involved in the query forces the selection process to consider tuple substitution as the only alternative because whether reduction can be used and what the components will be depends upon the structure of the query. Thus it is possible to set up a model comparing the costs of different substitution variables in the following manner. Let C_i be the cost of processing the query if variable i is substituted. Then

$$C_i = t_i + \sum_{\beta \in R_i} C(Q_\beta'(X_1,\dots,X_{i-1},X_{i+1},\dots,X_n))$$

which is the cost of substituting for X_i plus the cost of processing the remaining query once for each tuple in R_i. Now $C(Q_\beta'(X_1,\dots,X_{i-1},X_{i+1},\dots,X_n))$ must be estimated. If this cost is assumed independent of β, it can be approximated by $t_1t_2\cdots t_{i-1}t_{i+1}\cdots t_n$. This approximation ignores the fact that the cost of the last step will be in pages since it is not known which variable will be remaining at the end. Using these assumptions,

$$C_i = t_i + t_i(t_1\cdots t_{i-1}t_{i+1}\cdots t_n)$$

Thus the minimum C_i will correspond to the minimum t_i.

Obviously this criterion does not consider the effect of substituting for X_i on the structure and complexity of the remaining query. But, there is no way, just using size to take this into account. However it must be considered that cardinality is a readily available quantity and the decision process is simple. As a first step, this heuristic is not bad and since size is an important factor in determining the processing cost, it will certainly be used in some way in any selection process. But, by itself, it cannot claim to be even near-optimal.

Note that this criterion makes the choice of using reduction a separate decision. The variable is selected considering only substitution with the premise that if reduction can be used, it will do even better. This is not always a valid premise.

7.4 Size and Query Structure Criterion

The structure of the query obviously plays an important role in determining the processing costs. It determines what combinations of relations must be examined and thereby dictates the minimum product terms in the cost function. By using simply size and the structure of the query it should be possible to make a decision which will result in a near-minimal processing cost. Most of the observations concerning how the structure affects the cost are qualitative at this point but are still useful in selecting a "good" variable.

Consider first that when trying to estimate the effect of the current decision, the processing of the query can be examined all the way to completion. But this means considering a very large number of possible paths. Thus, the estimates which will be used only take into account the immediate effect of a decision and then use a perhaps crude estimate of the cost of processing the remaining query, not considering the many possible ways of answering that remainder.

Let Q be a query in variables $X_1,\cdots,X_n$ with ranges $R_1,\cdots,R_n$, $n>2$. Let us examine the cost of processing Q if X_i is the variable selected "for substitution". First, if X_i is actually substituted for, this will mean that every clause which originally involved X_i will now have one less variable. If any of the original clauses were two-variable with X_i, they are now one-variable clauses and can be executed to reduce the range size of one of the remaining variables. Thus, the size of the cross product which is to be examined will also be reduced. So, generally speaking, the more one-variable clauses which are a result of the substitution, the more the restriction which is gained. This can also be used in a negative sense. That is, when considering the substitution of a particular variable, if no restrictions are generated, it is usually not a good choice.

There is a particular class of one-variable restrictions which are beneficial to recognize. These are disjoint one-variable clauses. If such a clause appears, it is only necessary to verify the existence of a tuple which satisfies the condition and that range will not have to be included in any further processing. It can be shown that if substitution for a variable results in the query breaking apart into a series of disjoint one-variable subqueries, substitution for that variable will result in a cost less than substitution for any other variable, regardless of the respective range sizes. Thus, for a query such as

```
RETRIEVE ($X_1$.a)
   WHERE $X_1$.b=$X_2$.b AND $X_1$.c=$X_3$.c
```

X_1 should be selected considering only substitution. And since X_1 is a joining variable in such a situation, substitution for X_1 does better than reduction on X_1. Thus, this substitution will result in the minimum cost.

Note also that substitution for a joining variable is guaranteed to split the remaining query into disjoint pieces. Even though they may not all be one-variable pieces, this will break the cost into a sum of smaller products rather than one large product. Thus it is frequently more efficient to substitute for a joining variable instead of a non-joining one.

It is possible to compare the costs using reduction by determining what the joining variables are. Then reducing on each joining variable results in a set of components for each one. The cost for each reduction can be estimated as the sum of the product of the set of variables in each component. Then the reduction and its associated joining variable with the minimum cost can be selected. Now compare this result using only reduction with the selection made considering only substitution. Let $\{Q_R\}$ designate the set of components which result in the minimum reduction cost and let X_R be the associated joining variable. Let X_i be the variable selected when considering only substitution. The query can be reduced into a set of components $\{Q_S\}$ given X_i using the rules presented in Chapter 5.

If $\{Q_S\}$ and $\{Q_R\}$ are the same set of components, there is no problem. By using the conclusions of Chapter 5 it can be determined whether reduction or substitution should be performed.

When $\{Q_S\}$ and $\{Q_R\}$ are different, first examine X_i and $\{Q_S\}$. If the conclusions of Chapter 5 designate that reduction to $\{Q_S\}$ should be performed given X_i, then, since $\{Q_R\}$ results in the minimum reduction cost, reduction to $\{Q_R\}$ should be performed.

The only case remaining is when $\{Q_S\}$ and $\{Q_R\}$ are different and X_i should be substituted rather than reducing to $\{Q_S\}$. This implies, from Chapter 5, that either X_i is a joining variable or X_i is in the target list component of $\{Q_R\}$. Consider first when X_i is joining. It is known that substitution for X_R is better than reducing to $\{Q_R\}$. And since substitution for X_i is less expensive than substitution for X_R, substitution for X_i is more efficient than reducing to $\{Q_R\}$. When X_i is in the target list component of $\{Q_R\}$, to accurately compare the two under all conditions, statistics are required. However it is felt that reduction should be chosen for the following reasons. The effect of substitution for X_i is isolated to one component and thus can be realized all at once just as if substitution were performed now. But the remainder of the query which is not effected by X_i is split off and thus its cost will not be multiplied by the cost of substituting for X_i.

It is felt that by using these intuitive guidelines concerning the query´s structure, tempered by the relative range sizes, a near-optimal choice can be made for the current decision. Obviously when there is a variable whose range size is oustandingly small or large this will influence the selections suggested. But whenever there is not a large discrepancy among the range sizes, the structure of the query will be dominant and the above observations will hold.

7.5 Size, Query Structure and Storage Structure Criterion

When trying to decide what the next processing action should be for a query which involves three or more variables, the storage characteristics of the relations will not play as important a role as it does for two-variable queries. Since the effect of an action is only considered for one level, the remaining query will involve at least two variables and the storage structure is only useful at the one variable level. However there are certain intuitive observations which can be stated.

If one is trying to decide between two substitution variables with similar costs, if one of their ranges has an advantageous structure, the other variable should be selected for substitution purposes. This will allow the structured relation to remain available at the next level, which is perhaps a two-variable query where it can be used beneficially.

This points out that generally substitution over relations with useful storage characteristics should be delayed for as long as possible. But, this consideration should not be a dominant factor in the selection criterion above the two-variable level because the savings gained by using the access path at the bottom level may not be enough to overcome the extra cost incurred by using another more expensive choice at a higher level.

7.6 Selection Criterion Using Statistics

When statistics are available it is possible to combine the observations made previously in a qualitative sense into a more accurate quantitative cost estimating procedure. Define C_i as the cost of processing Q if X_i is selected. Then

$$C_i = t_i + \sum_{\beta \in R_i} C(Q_i(\beta))$$

if X_i is substituted tuple-by-tuple. Or

$$C_i = \sum_{q \in S_i} C(q)$$

if Q is reduced given X_i, where S_i is the sequence of subqueries Q is reduced to. Which of the costs will be used for each X_i will be governed by the conclusions of Chapter 5. Then we can define

$$C^* = \min_i \{C_i\}$$

The following discussion will develop means of estimating C_i so that C^* can be determined.

The first approach to estimating C_i is to consider only the cost if X_i is substituted. This assumes that the best variable for substitution is also the best variable for reduction. With this assumption, $C(Q_i(\beta))$ can then be taken as independent of β and i. This will result in selecting X_i with minimum range size. This policy was discussed in Section 7.3. However, since this does not take advantage of the statistics which are available, there seems no reason to keep the estimation this simple.

The next approach attempts to go one step further. First, determine if Q is reducible. If not, substitution is the only alternative. But, if Q is reducible, determine for each X_i if Q should be reduced by the rules outlined in Chapter 5. If so, this will reduce Q to a sequence S_i of subqueries. So we have

$$C_i = \sum_{q \lessdot S_i} C(q).$$

Now since this reflects the structure of Q, a relatively crude estimate could be used for $C(q)$. For example, we might take

$$C(q) = \prod_{j \in q} t_j$$

Now it is necessary to estimate the cost of substitution for X_i assuming Q should not be reduced for X_i. We have

$$C_i = t_i + \sum_{\beta \in R_i} C(Q_i(\beta))$$

A very crude estimate would be

$$C_i = t_i + \prod_{j} t_j$$

where $j = 1,2,\ldots,n$. This obviously does not consider the structure of Q at all and it seems fairly clear that if there was some X_i for which Q should be reduced, it would be selected over all X_j which should be substituted. Also, this method still does not take advantage of the statistics.

It is possible to consider if $Q_i(\beta)$ is reducible. But to make use of this fact, it would have to be assumed that for some X_j in $Q_i(\beta)$, reducing $Q_i(\beta)$ would be optimal. Even assuming that much, it is not clear exactly what the sequence of subqueries would be since this depends on the X_j. To carry the process further, that is estimating $C(Q_i(\beta))$ for each X_j in $Q_i(\beta)$, would quickly result in a combinatorial estimation problem. So, it does not seem reasonable to consider the reducibility of $Q_i(\beta)$. But there are certain structural characteristics which can be used.

First, substituting for X_i could have generated one-variable clauses. Using these results in

$$C_i = t_i + \sum_{\beta \in R_i} \left(\sum_{j \in J} P_j \right) + \prod_{k} t_k' \qquad (7.2)$$

where J is the set of indices for the ranges involved in the one-variable clauses and $t_k' = t_k$ if $k \notin J$. This equation is more accurate and since statistics are available, t_k' can be estimated.

Equation 7.2 can easily be made slightly more accurate by differentiating between the variables X_i, $i \in J$ which are now disjoint, i.e. those variables which appear only in one-variable clauses.

$$C_i = t_i + \sum_{\beta \in R_i} \left(\sum_{j \in J_1} P_j + \sum_{j \in J_2} p_j P_j \right) + \prod_{k \notin J_2} t_k'$$

where $J_1 = [j : X_j \in Q_i(\beta) - \overline{Q}_i(X_j)]$ and $J_2 = [j : Q_i(\beta) \cap \overline{Q}_i(X_j) = \emptyset]$ (disjoint).

The p_j's used in the equation are meant to represent the average percentage of R_j which will have to be examined to determine the existence of a qualifying tuple ($0 \leq p_j \leq 1$). The equation can be simplified by simply setting $p_j = 1$. However, if the statistics are available to estimate what

$t_k{}'$ would be, an estimate of p_j would also be possible.

It is also possible that substitution of X_i will cause Q to be split into several disjoint pieces. Since it is known that each piece will be processed separately,

$$C_i = t_i + t_i \sum_{q \in D_i} \prod_{k \in q} t_k$$

where D_i is the sequence of disjoint components. Obviously, the one-variable clause formulas can also be included in this cost if applicable.

By using the statistics to estimate the reduced range sizes involved in the remaining product, it is possible to obtain a reasonable estimate as to what the effect of substitution for a particular variable will be. In the estimate suggested above for the reduction cost, the statistics were not really used. Again, it is possible to estimate the effect of a restriction appearing in one of the components. But, if all one-variable clauses are preprocessed, no new ones will be generated through reduction. Statistics could be used to estimate the range size of the joining variable for each component. However, this means examining possibly several domains′ distributions and determining their joint effect on the joining variable′s range. This would also have to assume that each domain is independent and that a sort operation is included between consecutive components to remove duplicates.

It is felt that by just considering the separate combinations of variables achieved by reduction enough of the query´s structure is reflected to obtain a reasonable estimate. And adding the calculations conerning the range size of the joining variable would greatly complicate the decision process. Thus, the main usage of the statistics would be in estimating the reduced range sizes after a restriction is generated. They could also be used, as was mentioned in the previous section, to determine the effectiveness of a useful storage structure.

Another option available is to use sampling to obtain a cost estimate. If the number of variables in Q is small it might even be possible to push the estimation all the way down to one-variable queries, using a small sample for the relations in Q. Certain options at each level can be eliminated as contenders since it is probable the costs of different paths will vary greatly. By being very selective at each level about which paths to continue, the n! possible paths could be reduced to a manageable number.

However, even if sampling is used in this manner, it would in general not be wise to use the path selected as a fixed order of processing. The final path selected should be dependent upon β and the one obtained by sampling would either be for an "average" β or it would be a set of paths parameterized by β. The conclusion is that if sampling is used, no matter how far down the decision tree the estimation is pushed, the final result should only affect the selection made at the top of the current decision tree.

Certain final comments should be made regarding sampling. First, if the database being used is not very large, sampling would be quite expensive. Either the sample would be too small to accurately reflect the characteristics of the data or it would be almost the same size as the database itself. Neither option is acceptable.

Second, if sampling is to be an option, the only feasible way of using it cost effectively would be if the sample was always available. It is not reasonable to create a sample every time it is to be used. So, the sample would be created along with the database, but would only be updated occasionally. In this way, the cost of creating and maintaining the sample would be associated with many uses of it, not just a single usage.

CHAPTER 8

EXPERIMENTAL RESULTS

The INGRES database system is an implemented, working relational database system. Due to this fact, it was possible to perform experiments to evaluate the hypotheses proposed in the previous chapters. The basic system used for this evaluation was version 6.0 of INGRES.

In evaluating these hypotheses we have taken as the cost function the number of data pages accessed. It is true that there are other costs involved in processing a query. However, many of these other costs, such as computing the target list function, will remain the same no matter what techniques are used for intermediate processing. Thus such costs can be ignored. In general, if the number of data page accesses is minimized, the overall processing cost for the query will also be minimized.

Using this criterion of evaluation, probes were inserted in the utilities process, the decomposition process and the one-variable-query-processor. These probes simply consisted of counters for measuring the number of pages accessed in the source and result relations for various types of queries. As such, the probes themselves have no effect on the measurements.

The decomposition process also was modified in the following respects to allow testing of all hypotheses:

1) a new substitution-variable selection routine was inserted to allow tuple substitution in a predetermined order.
2) a version was compiled which did not preprocess one-variable restrictions.
3) a version was compiled which allowed reformatting to be either always performed or never performed.

All of these changes caused only minor modifications in the standard decomposition process.

The results of the measurements will be presented in three sections. In the first section, the hypotheses concerning the usage of statistics in processing will be evaluated. Included here are topics such as preprocessing one-variable clauses and dynamically modifying the structure of relations. Even though statistics can play a very important role in variable selection it was felt that this step involves many other factors too and that the different selection criteria available should be presented together allowing for easier comparisons. So, variable selection criteria is the topic of the second section. Since the analysis of reduction was not able to reach any absolute conclusions for all cases, a number of experiments were performed to explore the many facets of reduction and to attempt to determine the real value of reduction as a processing tactic. The evaluation of reduction is presented in Section 3.

8.1 Hypotheses Concerning the Usage of Statistics

There are many areas within the processing algorithm where statistics can be used. In Chapter 6 several options which could take advantage of statistics were proposed and the effect upon the decisions made was analyzed assuming that full statistical knowledge of the domains was available. These steps included: 1) preprocessing one-variable restrictions, 2) projection prior to tuple substitution, 3) reformatting a relation to provide an efficient access path, and 4) dynamically creating a secondary index rather than modifying the data relation. Experiments were performed considering each of these options separately in an attempt to verify the theoretical analysis in practical application.

8.1.1 One-Variable Restrictions

One of the ways statistics on the data can be used is to determine, for each restriction, its effectiveness ratio (# tuples satisfying/total # tuples). The value of this ratio is then used to decide whether the restriction would be preprocessed. In Section 6.2 our analysis concluded that unless the ratio was very close to 1, the restriction should be performed. Thus, the knowledge of the distribution would not have an effect in the majority of cases.

To verify this conclusion, we processed several queries which called for a restricted product of two or more relations. A variety of restrictions were used in the queries in order to get a range of values for the effectiveness ratio. The sizes of the relations in the queries were also varied. In Tables 8.1 and 8.2 the measurements for two-variable and three-variable queries respectively are presented. C_i denotes the cost (in pages accessed) for the query when i of the available restrictions were preprocessed; this number includes the cost of processing the i restrictions. p_j denotes the effectiveness ratio for the restriction in variable j; $\bar{p}_j$ is the bound proposed by our earlier analysis. Thus, p_j must be less than $\bar{p}_j$ for executing the restriction to be cost-beneficial, according to the theoretical analysis. In all queries, variable 1 was selected for substitution first, then variable 2, etc.

In both tables, C_0 is by far the largest cost, even when p_i is very close to 1. In Table 8.1, there are three queries for which C_1 is the minimum cost rather than C_2 - queries 2, 4 and 11. For query 2, this is because $p_2 = 1.0$ so that even if the number of tuples was reduced by the restriction, they still occupy the same number of pages. Queries 4 and 11 both have a very small p_1, thus the benefit

TABLE 8.1. Performing one-variable restrictions in two-variable queries.

query	p_1	$\overline{p}_1$	p_2	$\overline{p}_2$	C_0	C_1	C_2
1	.957	.9995	.041	.9886	36432	34867	1910
2	.957	.9991	1.0	.9886	4140	3965	4009
3	.80	.9978	.226	.9967	20520	16464	4053
4	.426	.9978	.226	.9938	20520	8796	2207
5	.207	.9971	.944	.9474	1748	366	365
6	.10	.9979	.934	.9737	23560	2404	2313
7	.048	.9998	.207	.9891	1464870	70041	15507
8	.0096	.9998	.0047	.50	44726	436	226
9	.006	.9958	.583	.50	4290	44	46
10	.006	.9989	.4375	.50	16170	116	110

TABLE 8.2. Performing one-variable restrictions in three-variable queries. C_0 = 13,735,860 for all queries.

query	p_1	$\overline{p}_1$	p_2	$\overline{p}_2$	p_3	$\overline{p}_3$	C_1	C_2	C_3
1	.982	.999997	.9897	.999996	.262	.999995	13482879	13344176	3659292
2	.426	.999997	.551	.99994	.262	.99998	5855862	3224387	884358
3	.10	.999997	.106	.99996	.262	.99958	1373634	146211	40252
4	.011	.999997	.106	.99963	.262	.99597	144636	15481	4382
5	.003	.999997	.106	.9985	.262	.98387	36195	3946	1171
6	.003	.999997	.091	.9985	.262	.9811	36195	3388	1064

gained from the first restriction is very dominant. But note that for all these queries, C_2 and C_1 are very close; the extra cost of C_2 is not more than about 11%. And, if Table 8.2 is examined, the bounds $\bar{p}_i$ get even closer to 1 than in the two-variable case. For query 6, even though p_1 and p_2 both are less than .1, $\bar{p}_3$ = .9811 which is reasonably large.

These queries were run on relations varying from 92 to 3330 tuples and 5 to 758 pages. If these sizes get larger or when the number of variables is three or more, the advantages of preprocessing the restrictions will be even greater.

All of these measurements were performed on relations which had non-keyed structures. Thus every tuple had to be examined. Clearly if the relation is keyed on the domain(s) in the one-variable clause, the cost of performing that restriction would decrease. If that relation is later selected for tuple substitution, this will be the only effect on the results. However if that relation is not selected for substitution, then OVQP will be accessing it at some later point. If the restriction is not preprocessed, then that relation will only have to be accessed once to perform both the restriction and the final query. And this access can made using the keyed structure. When the restriction is preprocessed though, the result relation defaults to a non-keyed structure. So we have the option of leaving it non-keyed and thus having to access all pages or modifying the structure to one which can be used by the remaining query. But, this is a completely different issue than the one being examined here and the option of delaying the preprocessing of restrictions was not considered.

8.1.2 Projection Prior to Substitution

The intuitive reasoning behind this option is quite obvious. Tuple substitution is the most costly step in the decomposition algorithm. If the number of tuples which must be substituted can be reduced by removing duplicates, it would appear a profitable operation. The analysis supports this. Again, a bound was obtained on the percentage of non-duplicate tuples which must be found for the operation to be advantageous. This bound is very close to 1.

The queries run for measurement were all two-variable queries since it can easily be seen how the results generalize. The parameters varied were the percentage of non-duplicate tuples which must be substituted and the pages in the relation remaining after substitution. The results are separated into two cases. First, in Table 8.3, the case when a complete projection must be performed, i.e. the referenced columns selected and a sort done to remove duplicates. And, second, in Table 8.4, when the selection of

TABLE 8.4. Removal of duplicates prior to substitution.

q		$\bar{q}$	P_2	C	C_p
.19108 =	60/314	.99459	18	5966	1162
.99342 =	302/304	.99630	53	16416	16390
.26316 =	5/19	.99751	53	1026	275
.99013 =	301/304	.99048	20	6384	6370
.26316 =	5/19	.98965	12	247	70
.18421 =	7/38	.99286	27	1064	202
.71591 =	63/88	.99303	16	1496	519

The interesting thing to note here is the high values associated with $\bar{q}$. In all of the test queries, the number of original tuples and particularly P_2 were fairly small and yet the values of $\bar{q}$ are quite high. When these values of P_2 and original tuples are larger, the bound $\bar{q}$ will become quite close to 1.

8.1.3 Reformatting

This option was a very controversial one within the INGRES project. Intuitively, it appears that the cost of modifying a whole relation might be too high a price for the benefits obtained. However the analysis concluded that whenever the query on the reformatted relation is to be executed many times (at least 100 times), reformatting can be advantageous.

No experiment was done for the first case studied in the analysis of reformatting where a modify is not performed but only the referenced domains are selected from the original relation. The conclusions reached for that case are straightforward and present no questions in understanding or implementing.

Several queries were run where reformatting was used. The parameters varied include the number of substitutions to be performed, the distribution of the key domains and thus the percentage of pages which must be accessed using the keyed structure, and the reduction in width. In Table 8.5 the measurements for reformatting to a hashed structure are presented and Table 8.6 contains the results for modifying to an ISAM structure. For these tables, t_1 is the number of substitutions to be performed (and thus the number of times the one-variable subquery will be executed), c_2'/c_2 is the ratio of new page capacity to the original, p_2 is the expected percentage of pages which must be accessed for each substituted value to answer the one-variable subquery. p_2 must be less than $\bar{p}_2$ for the reformatting to be affordable according to the analysis. Another measure developed in the case of modify to hash is $\bar{t}_1$ which is the lower bound on the number of substitutions to be performed. C_r is the cost of processing the whole query using reformatting, including the cost of the copy and modify operations, and C is the cost if reformatting is not performed; both costs are in units of pages accessed.

The interesting thing to note in the first table is the correlation between $\bar{t}_1$, $\bar{p}_2$ and c_2'/c_2. As the ratio of page capacity increases, both of the bounds $\bar{t}_1$ and $\bar{p}_2$ are relaxed. In other words, $\bar{p}_2$ increases and $\bar{t}_1$ decreases. Specifically note that whenever $t_1 \geq 100$, if $c_2'/c_2 > 1$, then $\bar{p}_2 > 1$, and thus the constraint is always satisfied. For smaller values of t_1, as would be expected, the cases must be examined more closely. If the actual values of p_2 are examined, in most cases using a bound of $p_2 \simeq .2$ to calculate $\bar{t}_1$ is quite conservative. The main cases for which statistics would be helpful would be to predict when p_2 will be large, or equivalently, when there are only a small number of unique key values.

TABLE 8.5. Reformatting to a hashed structure.

query	t_1	$\overline{t}_1$	c_2'/c_2	p_2	$\overline{p}_2$	C	C_r
1	380	26.57	2.636	.01842	2.4656	20520	1184
2	330	35.47	1.0	.16554	.91402	3960	1666
3	330	31.95	1.091	.04047	1.00475	17820	1999
4	330	42.51	1.689	.13889	1.49778	6270	1164
5	330	18.86	4.375	.05146	4.1364	16170	1976
6	162	32.89	1.0	.07326	.8376	8748	2101
7	100	26.57	2.636	.02038	1.9886	5400	703
8	92	78.88	1.0	.26725	.3141	1196	622
9	92	70.12	1.689	.20742	.5537	1196	451
10	87	78.88	1.0	.1485	.2747	1191	299
11	38	32.89	1.0	.7773	.3074	456	1008
12	38	28.46	1.727	.13695	.5833	1710	806

TABLE 8.6. Reformatting to an ISAM structure.

query	t_1	c_2'/c_2	p_2	$\bar{p}_2$	C	C_r
1	330	1.091	.05517	1.0611	17820	1955
2	330	4.375	.062375	4.3367	16170	1486
3	162	1.0	.05217	.9398	8748	1356
4	100	2.636	.19194	2.5259	5400	916
5	92	1.0	.452	.8982	1196	972
6	92	1.689	.24908	1.5854	1196	404
7	92	1.689	.824	1.5854	552	504
8	87	1.0	.31417	.8923	1191	500
9	38	1.727	.16327	.7415	1710	937

For every query run, reformatting to an ISAM structure was less expensive than no reformatting, even when there was no reduction in tuple width. The same observation also holds here about the correlation between c_2'/c_2 and $\bar{p}_2$, only more so. That is, even when the page capacity ratio is 1, the values of $\bar{p}_2$ are quite high. Since a majority of the queries run for testing in the ISAM case involved only equalities, it might seem that reformatting to ISAM would be more useful than reformatting to a hashed structure.

8.1.4 Dynamic Index Creation

The analysis of this option was not as successful as the other cases in that no definite conclusions were reached. The only result was that distributional information would be very helpful in determining the effect of not clustering the values. It was hoped that performing some experiments would give a better indication of the costs.

The current INGRES system only supports using indices in a limited way. That is, when an index is available, an access is made to the index to retrieve the identifier of a qualifying data tuple. That data tuple is retrieved and then the index is examined again to find the next qualifying tuple. Also, the way that the tuple identifier is stored in the index does not allow ordering by data page number. If all identifiers of qualifying tuples are retrieved and then sorted by page number or even if the index is also ordered by page number, the use of indices would be more efficient.

Table 8.7 presents the results of the measurements in a comparative fashion. That is, the costs of modifying to hash, ISAM or creating an index are all included to gain some insight into their relationships. The first column in the table references the graphs following for the distribution of the key domains (Figure 8.1a-h). The next three columns are the costs involved in making the access path available - i.e., modifying the relation to hash (H) or ISAM (I), or creating the index (SI), which includes building it and modifying it to an ISAM structure. Then the following three columns are the costs of processing the query using the three options. These costs include the cost of providing the access path. All costs are in terms of data pages accessed. The last two columns give the ratio of page capacity for the modified relation to the original or the index to the original relation. Note that each index tuple contains the key domains and a 4 byte tuple identifier.

It can be seen from Table 8.7 that whenever reformatting includes a reduction in tuple width ($c'/c > 1$), creating an index is not a winning policy. Obviously if the tuple width of the index is the same or larger than the original relation, reformatting should be performed rather than indexing. In general however, the cost of creating the index and modifying it to an ISAM structure is less than modifying the structure of the original relation as was expected.

The following observations can be made from examining the distributions of the key domains:

1) hashing seems to do better when there are a small number of distinct key values and only a few of these key values account for most of the occurrences.

2) an ISAM structure, either for the primary relation or used for an index, is to be preferred when there is a fairly uniform distribution of occurrences vs. key values.

These conclusions are not unexpected. Notice however that the domains which are favorable to indexing have a distribution very similar to those for which reformatting to an ISAM

TABLE 8.7. Dynamic index creation compared to reformatting.

graph	H	I	SI	C_H	C_I	C_{SI}	c′/c	c_{SI}/c
f	889	760	207	1116	1203	619	1.0	2.36
f	1310	746	334	2101	1356	1026	1.0	3.0
g	645	1119	568	970	1495	7615	1.0	2.0
b	235	381	156	622	972	10478	1.0	2.138
b&h	151	63	64	345	433	620	1.0	1.207
g	539	626	568	806	937	7654	1.727	3.0
h	57	85	73	299	500	1204	1.0	1.414
h	236	84	60	391	344	1037	1.0	1.069
a	794	495	269	1721	1504	2535	4.375	2.75
c	495	150	334	703	536	2181	2.636	3.0
d	130	37	156	451	404	10478	1.689	2.138
e	772	127	233	1666	1504	2184	4.375	.9428
c	-	75	100	-	916	22595	2.636	1.14
d	-	25	96	-	504	10344	1.689	1.265
a	961	660	334	1999	1955	2789	1.091	3.0
e	831	168	233	1976	1486	2255	4.375	4.125

structure is preferable. However in the queries where indexing was better, for over half of the substituted values, no tuples in the accessed range satisfied. Whereas the queries favoring the ISAM structure had less than one-fourth of the substituted values without qualifying tuples.

So the conclusions seem to indicate that there are four characteristics which must be present in order for creating an index to be the preferred policy: 1) tuple width of the index must be less than that of the original relation, 2) reformatting the relation causes no reduction in tuple width, 3) there is a fairly uniform distribution of occurrences vs. key values, preferably only one or two occurrences per value, and 4) there is a relatively high percentage of substituted values for which there are no qualifying tuples.

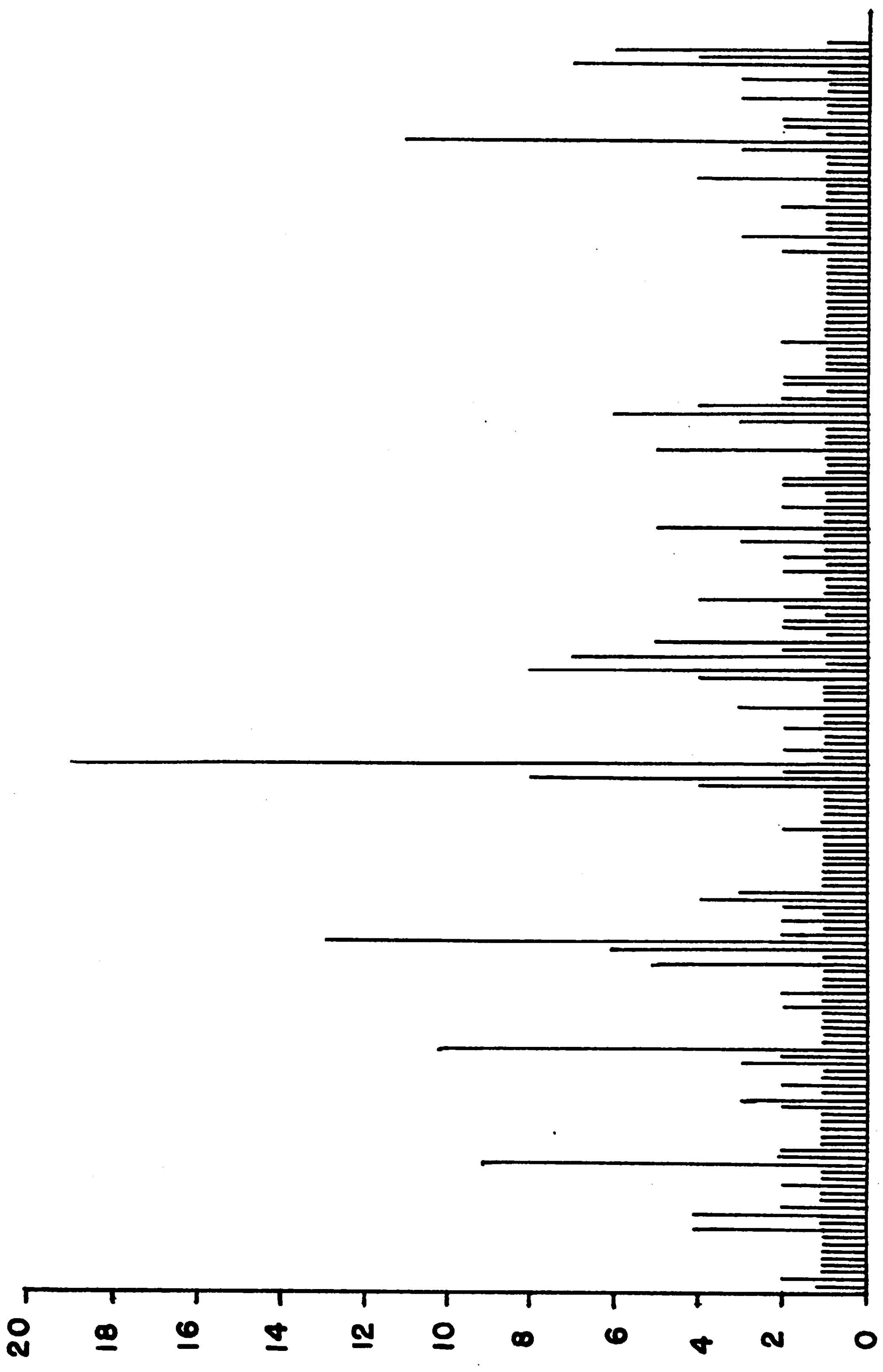


FIGURE 8.1. (a) 175 distinct values.

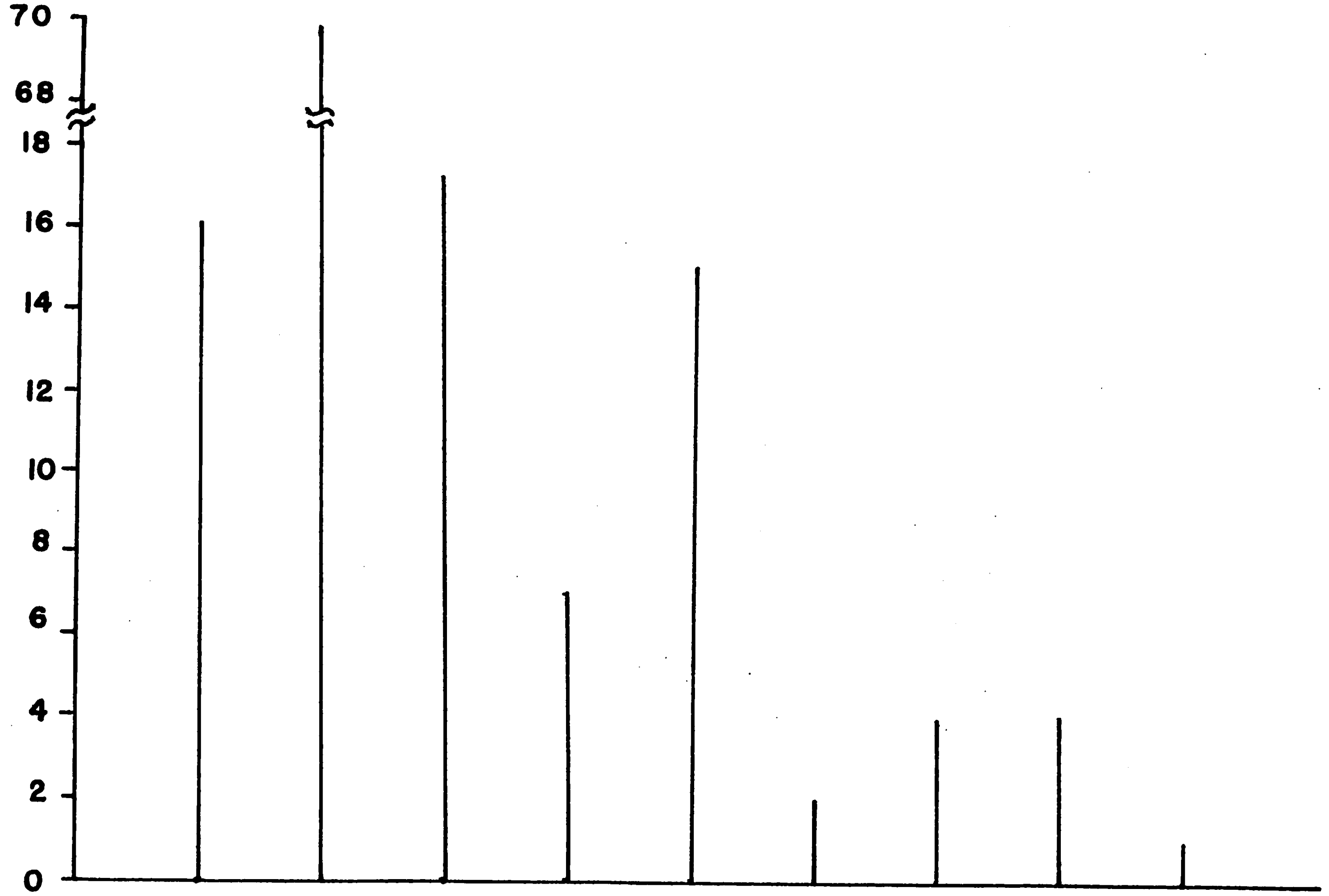


FIGURE 8.1. (b) 9 distinct values.

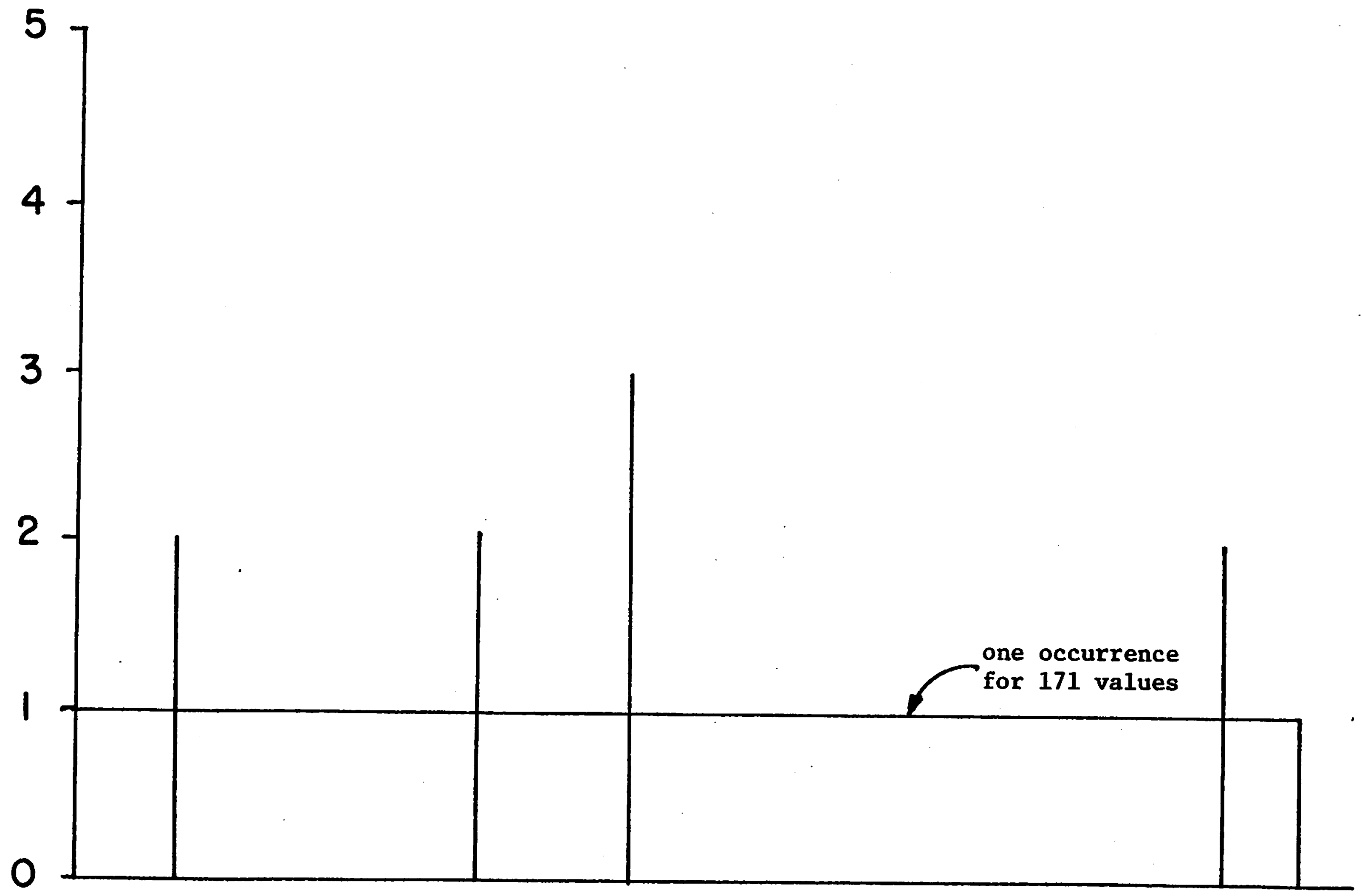


FIGURE 8.1. (c) 175 distinct values.

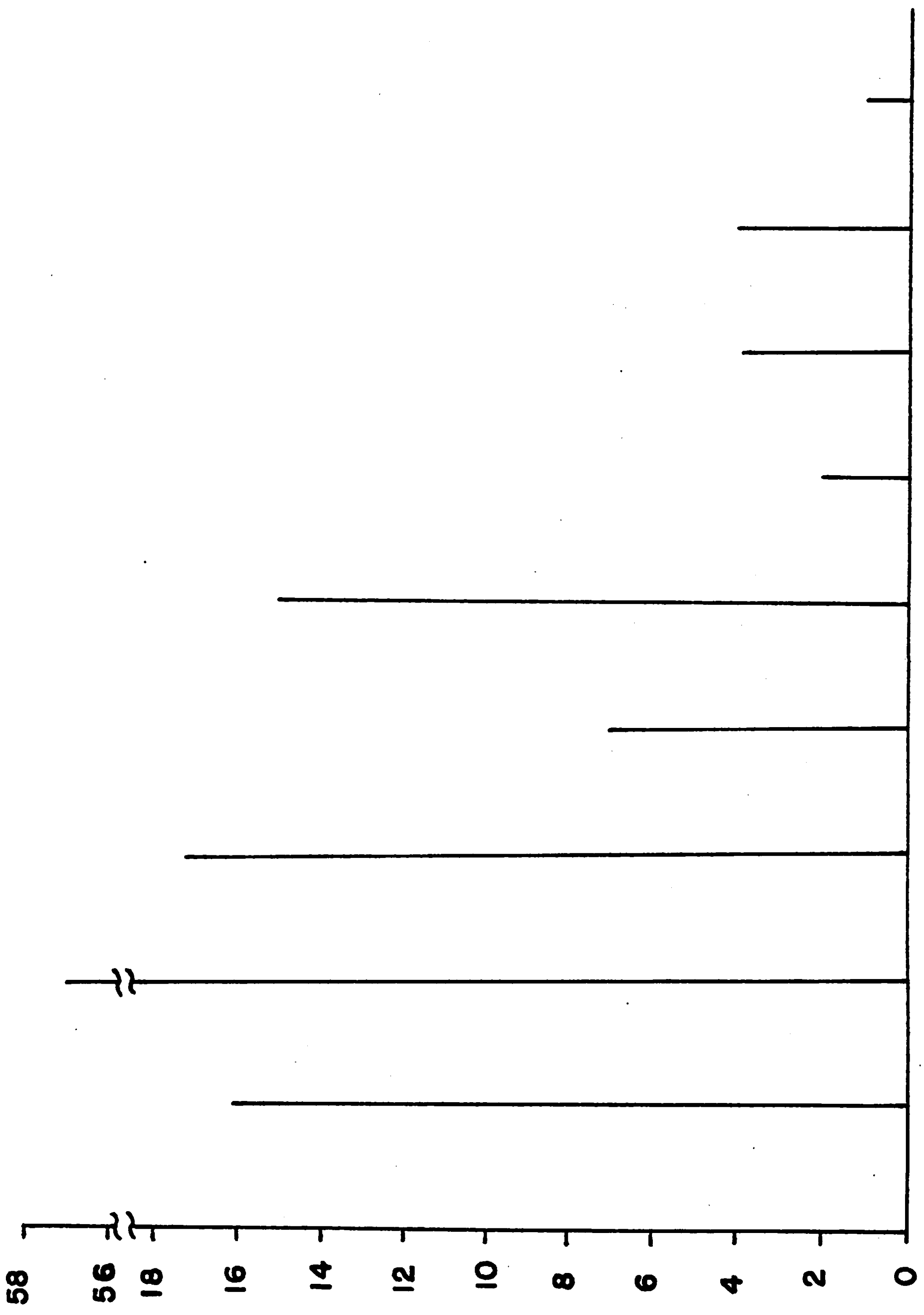


FIGURE 8.1. (d) 9 distinct values.

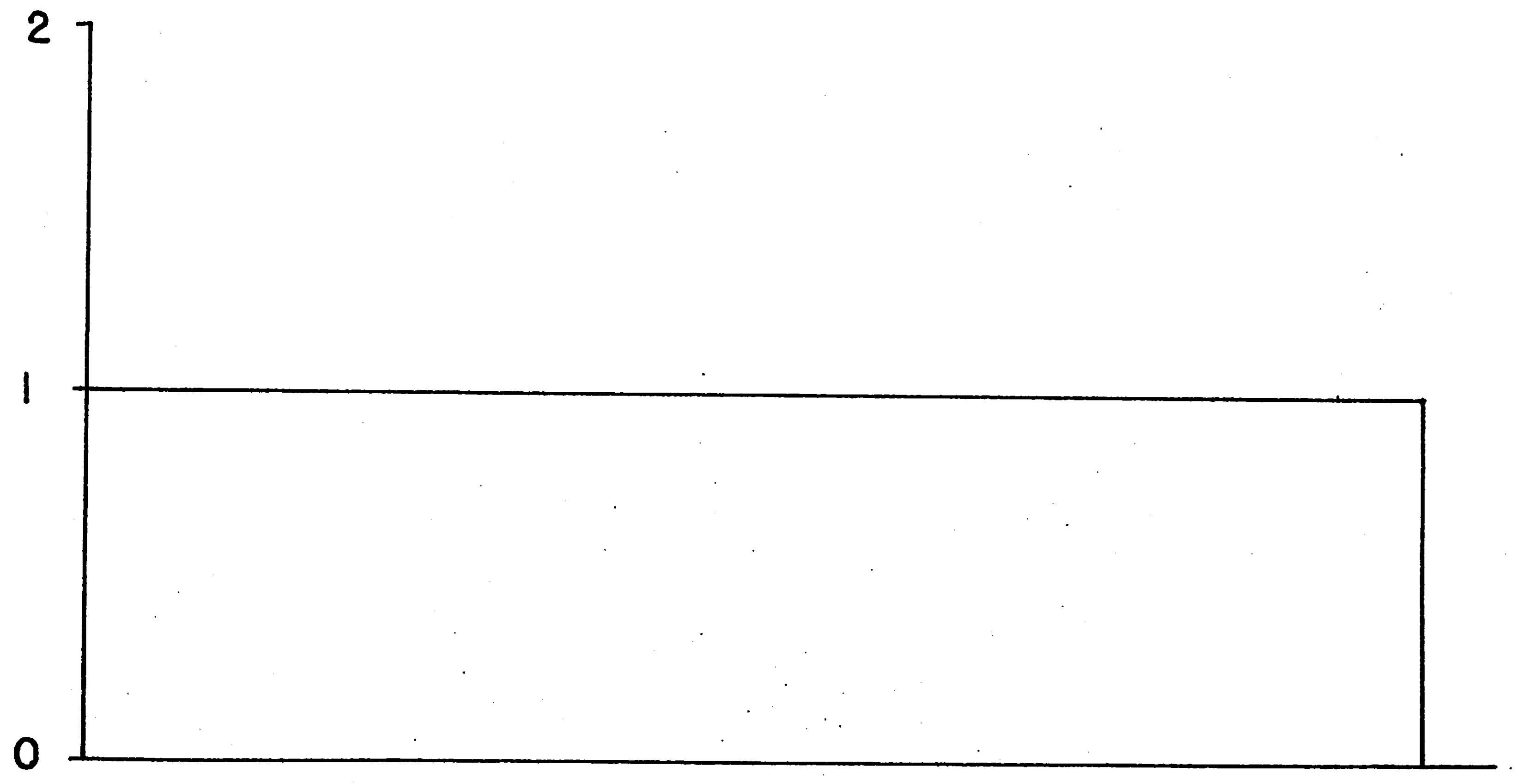


FIGURE 8.1. (e) one occurrence for 378 distinct values.

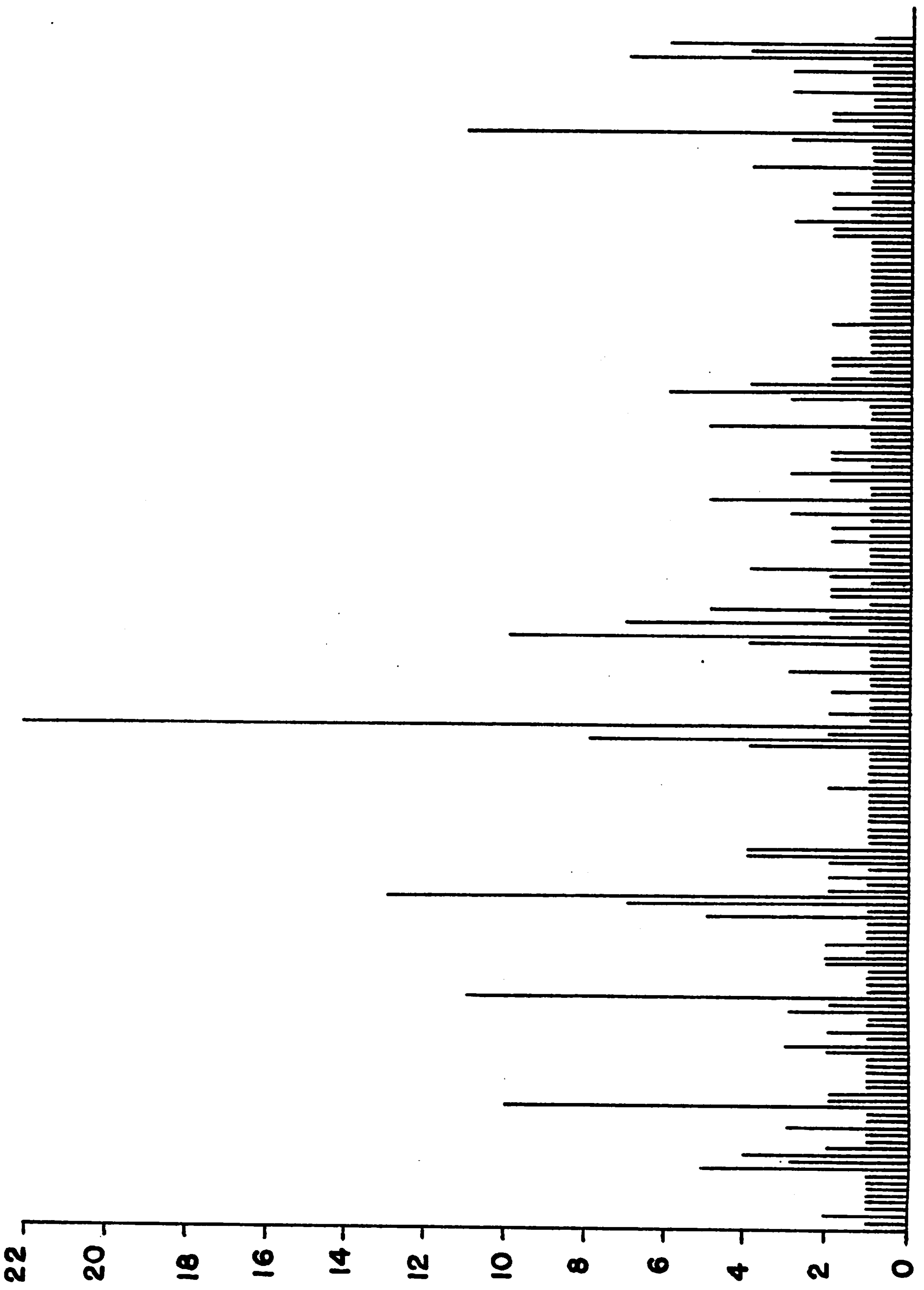


FIGURE 8.1. (f) 175 distinct values.

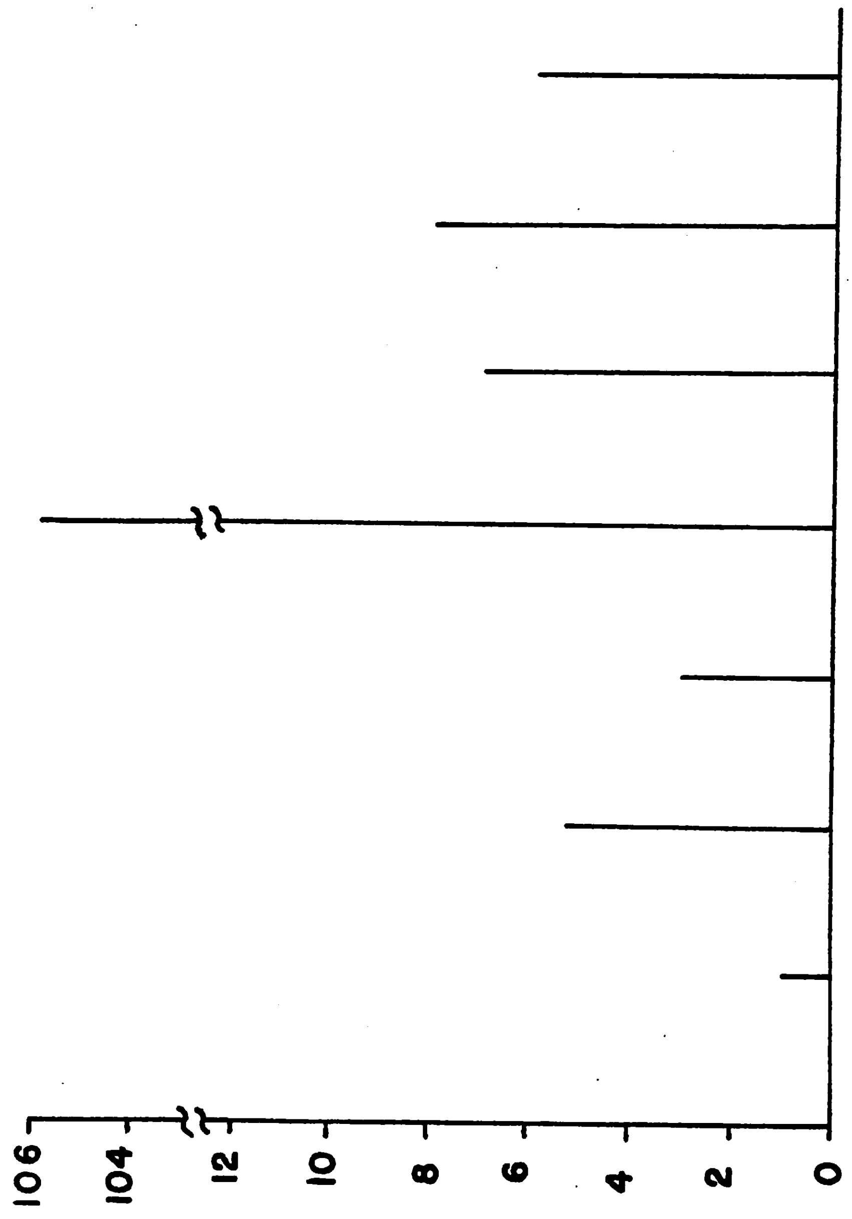


FIGURE 8.1. (g) 7 distinct values.

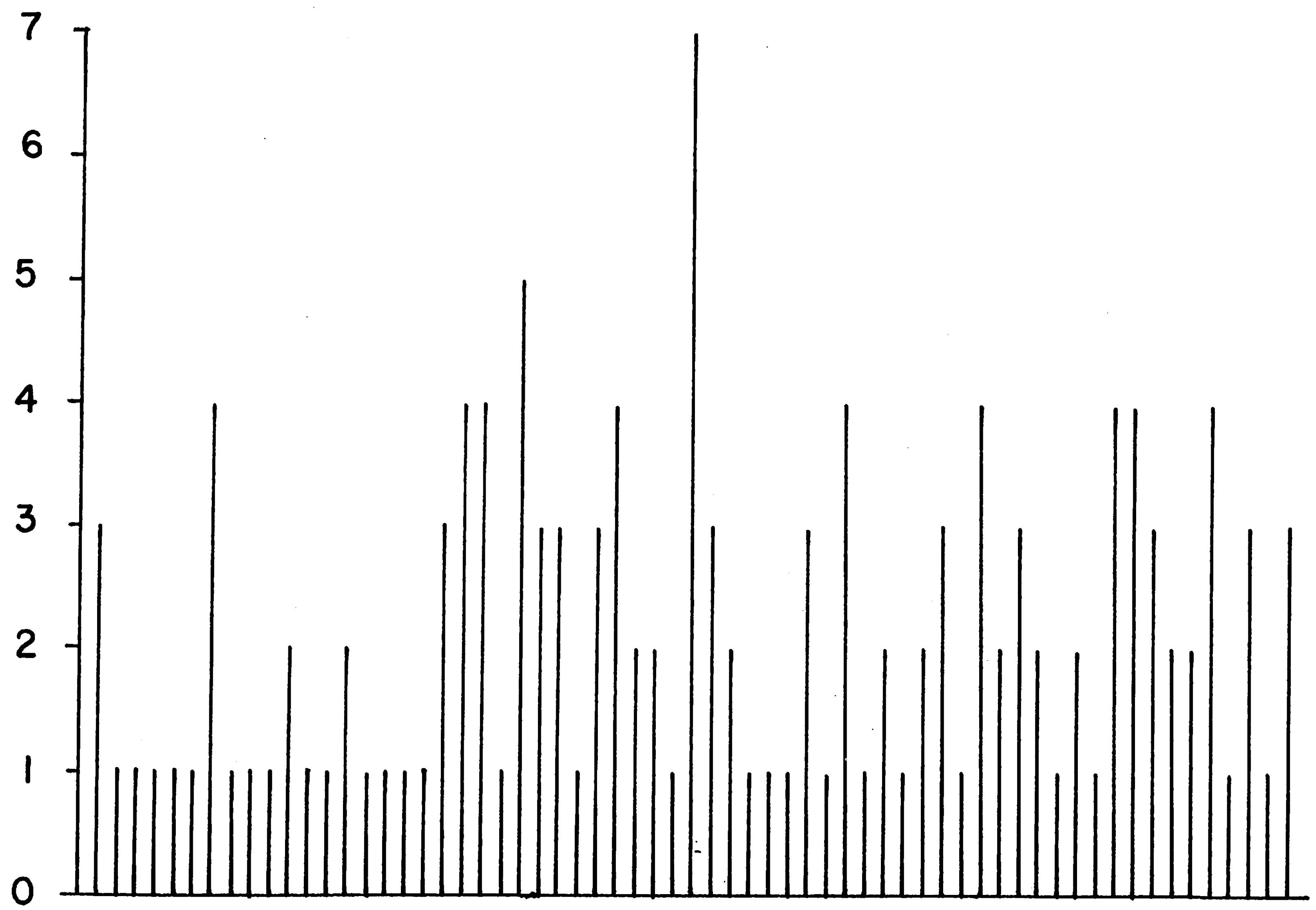


FIGURE 8.1. (h) 63 distinct values.

8.2 Variable Selection

This section will present the results of experiments run to test the hypotheses proposed in Chapter 7 on selecting a variable for tuple substitution. The experiments were broken down into two major groups - those for two-variable queries and measurements for queries involving three or more variables. Within each group, the query characteristics mentioned in Chapter 7 were varied and their effect on the processing cost considered. The first subsection contains the results of these experiments for two-variable queries and then, in the second subsection, three or more variable queries are discussed.

8.2.1 Two-Variable Queries

The following hypotheses were tested:

1) If size alone is considered, the variable which minimizes the ratio $\frac{t}{P+1}$ should be selected, where t is the cardinality and P is the number of pages for a relation.

2) The only useful query structure characteristic is the number of variables in the target list. In general, if there is only one, that variable should be selected for substitution.

3) If only one of the relations has a keyed structure which can be used, the other variable should be

selected for substitution to allow OVQP to take advantage of the access path.

4) If statistics are available, a cost estimate should be made considering all of the above characteristics of the query environment and the distribution of the linking domains.

Not only were each of these criterion tested separately to evaluate their credibility, but their effect was considered to determine their overall importance. This type of evaluation will help to define the order in which the criteria should be applied to a query to result in a minimum processing path.

In order to evaluate the first hypothesis, several queries were run varying the number of tuples and pages in each relation. The queries all essentially performed a join between the two relations and thus were of the form:

```
RETRIEVE (X1.a, X2.a)
    WHERE  X1.b = X2.b
```

Both relations had non-keyed structures for this phase of the testing.

The costs are in units of pages accessed by decomposition for substitution purposes and OVQP for answering the set of one-variable queries. $C_{1,2}$ is the cost of substituting for variable 1, while $C_{2,1}$ is the cost when variable 2 is substituted.

As can be seen from Table 8.8, the criterion of minimizing the ratio $\frac{t}{P+1}$ accurately selects the variable with the minimum processing cost. Notice that using only the number of tuples or only pages does not work as well. Thus the first hypothesis for the case when size alone is used appears true from the experimental results. Note that as the two ratios approach equality, the costs become more similar.

For testing the second hypothesis concerning query structure, several queries of the form

```
RETRIEVE (X1.a)
        WHERE  X1.b = X2.b
```

were run, varying the sizes and domain distributions involved. Again for this phase, non-keyed structures were used. In the following table (Table 8.9), the results of the experiments are presented. $C_{1,2}$ and $C_{2,1}$ are as described for Table 8.8. Variable 1 is always the variable which appeared in the target list.

For all queries except 2 and 8, the hypothesis is shown to be valid even though variable 2 would be selected using the size criterion. The reason this happens is that there is an implied existential quantifier on the variable not appearing in the target list. Once substitution is made for the target-list variable, the range relation of the remaining variable only needs to be examined until the first qual-

TABLE 8.8. Various orderings for substitution in two-variable join queries.

query	t_1	P_1	t_2	P_2	t_1/P_1+1	t_2/P_2+1	$C_{1,2}$	$C_{2,1}$
1	138	70	66	15	1.944	4.125	2104	4633
2	62	17	138	70	3.444	1.944	4307	2355
3	66	15	330	56	4.125	5.789	3709	4918
4	62	17	112	57	3.444	1.931	3538	1950
5	66	15	79	17	4.125	4.389	1135	1194
6	82	17	79	16	4.556	4.647	1329	1358
7	82	22	81	7	3.565	10.125	595	1788
8	62	31	380	32	1.9375	11.515	2014	11811

TABLE 8.9. Two-variable structured queries.

query	t_1	P_1	t_2	P_2	t_1/P_1+1	t_2/P_2+1	$C_{1,2}$	$C_{2,1}$
1	66	15	138	70	4.125	1.944	145	2104
2	62	17	138	70	3.444	1.944	3922	2355
3	330	56	66	15	5.789	4.125	1759	3709
4	62	17	112	57	3.444	1.931	733	1950
5	79	17	66	15	4.389	4.125	441	1135
6	62	32	56	29	1.879	1.867	154	1819
7	56	13	51	14	4.0	3.40	602	646
8	150	39	56	29	3.75	1.867	4017	2175

ifying tuple is found. In most cases, the resulting savings will be the dominant factor in variable selection. The reason that queries 2 and 8 do not follow this pattern is due to the distribution of the linking domain. For a large number of values of that domain in relation 1 there are no matching occurrences in relation 2, so the whole relation must be examined. When this happens, the savings made possible by the query structure are diminished due to the distribution. This detrimental effect can only be predicted if the distributions are known.

The results in Table 8.9 strongly suggest that when there is only one target-list variable it should be selected for substitution, especially when no statistical information is available. The potential savings are large and the occasional losses are limited. There is another advantage to this policy. Since OVQP simply retrieves all tuples that satisfy and appends them to the result relation without checking for duplicates, the potential size of the result relation is the product of the number of tuples of each relation. When the target-list variable is substituted first however, the result relation is bounded by the size of its range instead of the product of the two range sizes. This will have a beneficial effect on the cost if the result relation is subsequently modified to remove duplicates or if the result of this query is the first in a series.

The third hypothesis is very important for two-variable queries because this allows OVQP to take full advantage of the storage structure of the relations. This criterion should be used in combination with the size and query structure policies discussed above. Clearly, if variable 1 is selected for substitution using, for example the size criterion, and variable 2 has a keyed range relation, considering the storage structure will have no effect on the selection because the preferred ordering has already been chosen. However if variable 1′s range has the keyed structure then there could be some effect. For this reason, the experiments run only considered the effect of a keyed structure for the range of a variable which would be selected for substitution using either (a) the size criterion, or (b) the query structure criterion. The following table will include the minimum cost for each query from either Table 8.8 or 8.9 so that a comparison can be made to the cost using the decision policy of hypothesis 3. C will denote the minimum cost using non-keyed structures and K the costs measured considering the available keyed structure.

TABLE 8.10. Two-variable queries using a keyed storage structure.

query	C	K
a1	2104	3511
a2	2355	143
a3	3709	3764
a4	1950	2728
a5	1135	773
a6	1329	1105
a7	595	86
a8	2014	524
a9	1329	205
b1	145	1631
b2	3922	196
b3	1759	2785
b4	733	1501
b5	441	783

From these results it can be seen that hypothesis 3 is not in general true. For queries where the variable selection is made using the query structure (b1-5) in fact, the policy suggested by this hypothesis is almost always the wrong one. The only exception is query b2 which was a query where the selection using the query structure alone resulted in the greater cost. So its savings using a keyed structure is understandable.

For queries a1-9 where the selection was made on a basis of size, the results are varied and the benefit of the keyed structure depends on the distribution of the key. In approximately two-thirds of the cases, though, a substantial savings was achieved by deciding in favor of the keyed structure.

The fourth hypothesis is a fairly obvious one and clearly whenever statistics are available, an estimation procedure will predict the better path. From the previous discussions, the knowledge of the distribution would solve many questionable decisions. Unfortunately, INGRES does not support the capability of either maintaining or using statistical information currently so it was not possible to actually perform experiments to test this procedure. However, for several of the queries, the estimating procedure was carried out by hand. In all cases, the order selected was the one with the minimum processing cost when the query was run.

The results of the experiments and the analysis for selection of a substitution variable can be summarized as follows. Given a two-variable query, these factors should be considered in this order:

1) If distributional information is available, a cost estimate function should be applied to determine the processing path which accesses a minimum number of data pages.

When statistics are not available,

2) If the target list contains only a single variable, that variable should be selected for substitution regardless of size or storage structure.
3) When the target list contains two-variables, if one

of those variable has a keyed structure on the domain(s) in the qualification, select the other variable for tuple substitution.

4) If none of the above criteria applies, select the variable for substitution which minimizes the ratio $\frac{t}{P+1}$.

Note that these conclusions hold for all of the queries measured and will hold true in general. However, it is tacitly assumed that none of the relations are predominantly small ($\leq$ 10) or large. In these cases, size will undoubtedly be the dominant factor but, according to the previous conclusions, size is the last factor considered. What would be useful is some general measure which would reflect all of these considerations simultaneously.

One technique which can be used is to always consider the ratio t/(P+1) but to have P represent the "effective" pages of the relation. Using the actual number of pages the relation occupies is essentially assuming that every page will be accessed. When there is only a single target-list variable or a relation has a keyed structure, this assumption is not valid. Thus P can be modified to indicate the expected number of pages which will be accessed under these conditions. Clearly when statistics are available, it is a simple matter to estimate the effective P. When statistics are not available, it is still possible to use this technique.

When there is only a single target-list variable, it is always possible to create an index on the other relation and thus guarantee a single access per substituted value. Thus, P_{eff} for the qualification variable can be taken as 1. Another option is to assume that for 50% of the substituted values, all pages must be examined and for the remaining half, only a single page must be retrieved. Thus,

$$P_{eff} = \frac{.5t_s P + .5t_s}{t_s} = .5(P + 1)$$

where t_s is the number of tuples in the range of the target-list variable and P is the actual number of pages occupied by the other relation.

There has been considerable study done on determining the average number of page accesses required using a keyed structure [SEVE74, LUM71, HELD75c]. For a hashed structure, it is approximately one page access but this depends on the number of tuples per page. For a directory structure (ISAM), it depends on the range of values being retrieved. But, using these type of estimates for P_{eff}, the effect of the storage structure can be reflected.

Thus, for two-variable queries, the ratio $\frac{t}{P_{eff}+1}$ should be used as the selection criterion.

8.2.2 Multi-Variable Queries

For these experiments, a variety of three and four-variable queries were run. The structure of the queries, the sizes of the relations and the domains appearing in the qualifications (and thus the distributions) were all varied. Since Section 8.3 contains the evaluation of reduction as a processing alternative, this section will present the results considering only substitution as a processing tactic. As in the two-variable case, it was not possible to actually perform experiments to evaluate using statistics in a cost function as a prediction method. However, this procedure was carried out by hand and the measurements for the processing order selected by this method are included.

The following criteria for selecting a variable for substitution were compared:

a) select the variable whose range relation has the fewest tuples.

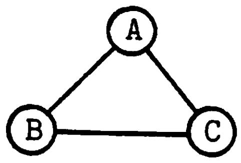
b) select the variable whose range characteristics minimize the ratio $\frac{t}{P+1}$.

c) select the variable which will have the greatest immediate effect on the remaining variables; that is, the variable which appears in the most two-variable clauses.

Although it is not a selection criterion as such, experiments were also performed to test the hypothesis that storage structure does not play a dominant role for queries involving three or more variables. To illustrate this, the queries were also run with keyed structures available in an attempt to see if the same order was the best whether or not the storage structure was considered.

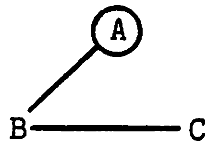
The results of the experiments will be presented in the following manner. First, a graph, as described in Chapter 5, for each query will be presented. Then the resulting costs in data pages accessed will be tabulated by query by criterion being evaluated and the variable selected for substitution will be indicated. Since there are several possible orderings for substitution starting with the same first variable, the cost tabulated is the minimum cost associated when the indicated variable is the first one selected for substitution. Also included in the last columns of Table 8.11 is the decision which would be made using complete statistics to estimate the various costs and the measured cost for that decision. The cost estimates using the statistics were calculated by hand.

Query 1



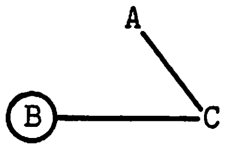
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 51 \quad P_C = 14$

Query 2



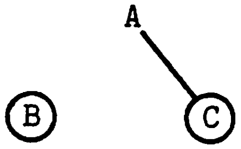
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 51 \quad P_C = 14$

Query 3



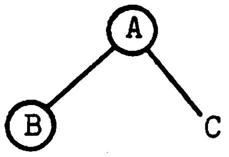
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 150 \quad P_C = 39$

Query 4



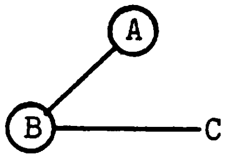
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 150 \quad P_C = 39$

Query 5



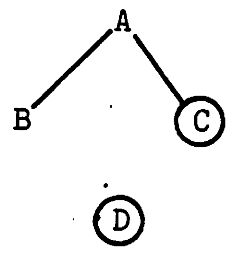
$t_A = 62 \quad P_A = 32$
$t_B = 62 \quad P_B = 17$
$t_C = 56 \quad P_C = 29$

Query 6



$t_A = 66 \quad P_A = 15$
$t_B = 51 \quad P_B = 14$
$t_C = 56 \quad P_C = 13$

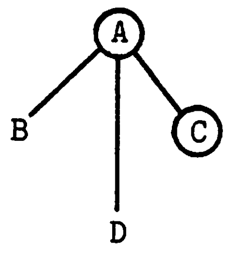
Query 7



$t_A = 98 \quad P_A = 50$
$t_B = 79 \quad P_B = 17$

$t_C = 62 \quad P_C = 17$
$t_D = 40 \quad P_D = 3$

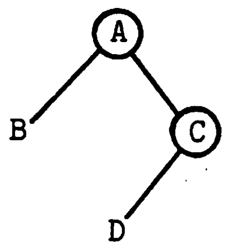
Query 8



$t_A = 56 \quad P_A = 29$
$t_B = 62 \quad P_B = 17$

$t_C = 112 \quad P_C = 57$
$t_D = 97 \quad P_D = 26$

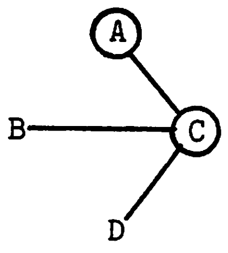
Query 9



$t_A = 62 \quad P_A = 32$
$t_B = 51 \quad P_B = 14$

$t_C = 56 \quad P_C = 29$
$t_D = 62 \quad P_D = 17$

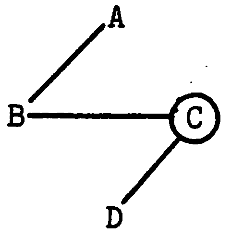
Query 10



$t_A = 62 \quad P_A = 32$
$t_B = 62 \quad P_B = 17$

$t_C = 138 \quad P_C = 70$
$t_D = 51 \quad P_D = 14$

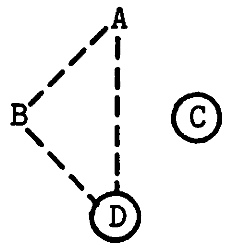
Query 11



$t_A = 62 \quad P_A = 17$
$t_B = 51 \quad P_B = 14$

$t_C = 62 \quad P_C = 32$
$t_D = 56 \quad P_D = 29$

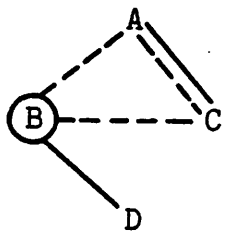
Query 12



$t_A = 51 \quad P_A = 14$
$t_B = 56 \quad P_B = 29$

$t_C = 62 \quad P_C = 17$
$t_D = 62 \quad P_D = 32$

Query 13



$t_A = 51 \quad P_A = 14$
$t_B = 56 \quad P_B = 29$

$t_C = 62 \quad P_C = 32$
$t_D = 62 \quad P_D = 17$

First, a few comments about Table 8.11. Query 1 has no interesting query structure - all three variables appear in the target list and each variable appears in two two-variable clauses. Thus, the query structure criterion is not applicable. For queries 9 and 12, there are two entries in the criterion (c) column. This is because there are two variables which possess the same query structure charac-

TABLE 8.11. Variable selection for multi-variable (n>2) queries.

Query	t_i/P_i+1				Criterion							
					(a) tuples		(b) t/P+1		(c) query		(d) stats	
No.	A	B	C	D	var	cost	var	cost	var	cost	var	cost
1	4.13	4.0	3.40	-	C	2159	C	2159	-	-	B	1291
2	4.13	4.0	3.40	-	C	3942	C	3942	B	806	B	806
3	4.13	4.0	3.75	-	B	2343	C	2343	C	2343	C	2343
4	4.13	4.0	3.75	-	B	144760	C	2261	C	2261	C	2261
5	1.88	3.40	1.87	-	C	13496	C	13496	A	1083	A	1083
6	4.13	3.40	4.0	-	B	1015	B	1015	B	1015	B	1015
7	1.92	4.40	3.40	10.0	D	128083	A	3015	A	3015	A	3015
8	1.87	3.40	1.93	3.59	A	3753	A	3753	A	3753	A	3753
9	1.88	3.40	1.87	3.44	B	28681	C	11266	C	11266	A	7152
									A	7152		
10	1.88	3.44	1.94	3.40	D	~72569991	A	113296	C	6249	C	6249
11	3.44	3.40	1.88	1.87	B	3367	D	33486	C	927	C	927
12	3.40	1.87	3.44	1.88	A	15778	B	15829	B	15829	B	15829
									D	14751		
13	3.40	1.87	1.88	3.44	A	4591	B	19815	B	19815	B	4591
									A	4591		
									C	19296		

teristics. So some other criterion must be used to decide between the two, but both are available.

The results presented in Table 8.11 seem to point to several conclusions:

1) Size in terms of number of tuples alone is generally not an adequate criterion to minimize the processing cost.
2) The structure of the query, specifically the number of two-variable clauses each variable appears in, is a dominant factor in determining the processing cost.
3) Using the ratio $\frac{t}{P+1}$ is a more successful criterion than using only t to select a substitution variable.

The only result which is perhaps unexpected is that the ratio of tuples to pages is better than tuples alone when considering only size to select a substitution variable. This implies that the assumption made in Chapter 7 that the last step of a multi-variable query being in pages could be ignored was not a valid assumption.

When the storage structures of the relations were modified so they were keyed on a useful domain, obviously, the total costs went down but the relationship between the costs using the three criteria was unaffected. This would seem to indicate that storage structure does not play an important role in the selection procedure when more than two variables are involved.

The conclusions of this criterion evaluation for selecting a variable for substitution in a query involving three or more variables can be stated as:

1) Obviously, if statistics are available, a cost function should be used to estimate the order of substitution with minimum cost.
2) For each variable, count the number of two-variable clauses in which it appears. Also, if the target list is one or two variable, this should be included in the count for the appropriate variables. Select the variable for substitution whose associated count is the largest. If there is a tie between two or more variables, use the next criterion as the deciding factor.
3) Select the variable whose range relation characteristics minimize the ratio $\frac{t}{P+1}$.

It should be noted at this point that in obtaining these results we did not consider reduction as a possible processing alternative. The effect of reduction on variable selection for tuple substitution will be considered in the next section.

8.3 Reduction

The idea of reducing a query into components which have only a single variable in common is an important part of this work and it is a new technique in the context of database processing. As such, there are many questions which arise concerning it. In Chapter 5, an attempt was made to answer one of the major questions. Namely, how useful is this technique as opposed to tuple substitution. But no definite conclusions were reached for all cases and it has yet to be shown how valid those results are in practice.

The ideal case for reduction is when the resulting sequence of components contains only two-variable queries because generally two-variable queries are less expensive to process than three or more variable queries. What can be learned by examining such cases? First of all, note that all components except the last, which contains the original target list, are guaranteed to have only a single variable in the target list, namely the joining variable. This fact will surely have an effect on the cost of processing. Since the cost will be a sum of costs for two-variable queries, is there a single component which will dominate the cost? This fact could be useful in predicting the total cost of processing and comparing this method to others.

When reduction is used, it is known that the result of an intermediate component will be used as a source relation for a subsequent component. One of the points in favor of a single-overlapping variable reduction is the fact that this result relation will usually be smaller than the original. Once can guarantee that it will be no larger by performing a sort operation to remove duplicates between intermediate components. But obviously this operation will have a cost associated with it. So the question to be answered is how beneficial is it to remove the duplicates between intermediate components.

One final question concerns the effect of keyed storage structures on reduction. The analysis performed in Chapter 5 considered that all relations possessed non-keyed structures so that all tuples had to be examined. This implied the assumption that a keyed structure would have no effect on the conclusions of reduction vs. tuple substitution. However, it was not shown that this was a valid assumption.

These are the basic questions which were considered when performing the evaluation of reduction. The case of reducing to all two-variable components and performing a sort between components is presented in the first subsection. Then the measurements concerning the hypotheses proposed in Chapter 5 for reduction vs. substitution are discussed in the second subsection. Also the effect of keyed structures is considered.

8.3.1 Reduction to Two-Variable Components

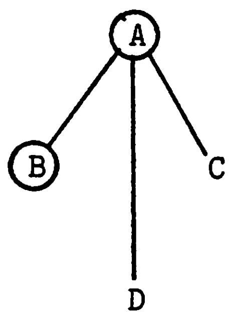
A series of three to five-variable queries were run to test the following hypotheses:

1) In every component except the last, which contains the original target list, the joining variable should be selected for substitution.
2) Either the first component executed or the last one will have the dominant cost compared to all other components. If the rule in hypothesis (1) is followed, it will generally be the last component which is dominant.
3) If the rule in hypothesis (1) is followed, the sorting cost will be minimal and the total cost of processing including the sort will be better than the cost without the sort.

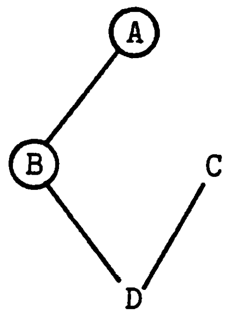
The second hypothesis is based on the following reasoning. The first component executed will generally have a large cost compared to the other components because there is no effect of the reduction yet. All subsequent components will involve a joining variable whose range has hopefully been reduced. The last component will usually involve examining a cross product whereas all previous components were guaranteed to have only a single target-list variable. Thus, its cost can conceivably be larger than the other components. This fact, if true, could be of use in predicting the cost of the entire query or in determining which component deserves the most attention for possible optimization.

Since the queries tested all reduce to two-variable components, the possible query structures are limited. The structures considered were of the following form:

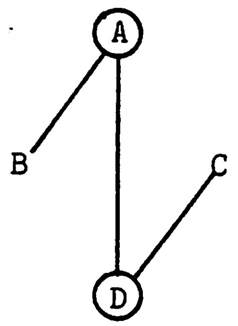
(a)



(b)



(c)



For queries of the form of (a) there are obviously different orderings for executing the components, i.e. 1 - A,C; 2 - A,D; 3 - A,B or 1 - A,D; 2 - A,C; 3 - A,B. It would be useful to be able to predict which ordering results in the minimum cost. Unfortunately the measurements did not provide any definitive answer to this question since in reality, it depends on the distribution of the domains involved. However, in general, it appears that performing the component which involves the relation with the smallest number of pages first is a good policy when statistics are not available. Thus, the measurements presented will be for the minimum cost ordering of components only.

For the other structures, little choice is available in the order of processing components. For form (b), the only order is 1 - C,D; 2 - B,D; 3 - A,B. For (c), there are two orderings: 1 - A,B; 2 - C,D; 3 - A,D or 1 - C,D; 2 - A,B; 3 - A,D. Since components 1 and 2 are not connected, the ordering will make no difference on the cost.

The sizes of the relations involved vary from 15 tuples, 2 pages to 138 tuples, 70 pages. All relations have non-keyed structures. Table 8.12 contains the results of the experiment. The letter in the structure column refers to the structures shown graphically above. Order 1 refers to the order of substitution within a component as described in the first hypothesis; that is, always substituting for the joining variable in the intermediate components. Order 2 is when the non-joining variable is chosen for substitution in the first component and then the joining variable in all remaining intermediate components. Tests were run where the order was also varied in other components, but the first component seems to be the critical one. Under the columns,

TABLE 8.12. Reduction to two-variable components.

Query No.	Graph	Order 1 w/sort	Order 1 w/o sort	Order 1 Size	Order 2 w/sort	Order 2 w/o sort	Order 2 Size
1	a	1044+25	3698	55/24	3710+1157	314102	4785/24
2	a	765+30	2505	55/24	2253+270	40145	1430/24
3	a	732+25	2359	55/24	2072+723	56052	2750/24
4	a	775+20	1169	62/38	2440+669	58625	3472/38
5	a	395+20	1226	55/13	1403+262	29228	1430/13
6	a	1021+25	3724	55/24	1305+25	4008	55/24
7	a	1037+57	3367	138/67	4274+1704	246798	6900/67
8	b	673+25	2573	51/4	1355+591	109086	2856/4
9	b	777+11	3558	50/1	1298+11	4078	50/1
10	b	724+20	7985	51/4	978+47	9899	204/4
11	b	865+25	5155	51/4	1547+591	108988	2856/4
12	c	1105+15	1159	15/3	2076+124	3585	750/3
13	c	1016+30	2709	15/3	1987+139	3680	750/3
14	c	1016+30	2709	55/23	2504+239	15311	1430/23

order w/sort, the numbers are of the form x + y. y refers to the cost of the sort operations while x is the cost of processing the query excluding the sort cost. The numbers in the size columns are the number of tuples before the sort/after the sort for the result of the first component.

In all cases, the total cost of processing the query is less when the sort operation is performed than when the duplicates are not removed, and the difference is usually large. Order 1, where the intermediate target list variables are chosen for substitution, is always preferable to order 2 whether the sort operation is included or not. This supports the conclusions of the two-variable selection criterion for structured queries.

One must realize that most of these queries reduce to three or four components, so the sort cost indicated is the sum of the sort costs after each component except the last. When the individual components are examined, it turns out that only the sort operation after the first component executed results in much reduction in the range size, except for queries with structure (c). For structure (c), there are essentially two "first" components since they have no variable in common.

Table 8.13 presents the individual component costs for each query and the size of the result relation before and after the sort operation. The costs used are those when

TABLE 8.13. Reduction to two-variable components, individual component costs.

Query No.	Graph	Component 1 Cost	Component 1 Size	Component 2 Cost	Component 2 Size	Component 3 Cost	Component 3 Size	Component 4 Cost
1	a	162	55/24	198	24/12	684	-	
2	a	149	55/24	195	24/13	37	12/12	384
3	a	150	55/24	198	24/12	384	-	
4	a	154	62/38	621	-			
5	a	140	55/13	157	13/13	62	12/12	36
6	a	139	55/24	198	24/12	684	-	
7	a	344	138/67	277	33/13	416	-	
8	b	114	51/4	38	3/1	29	55/12	384
9	b	364	50/1	29	56/12	384	-	
10	b	165	51/4	38	3/1	29	55/12	384
11	b	114	51/4	38	3/1	29	55/12	684
12	c	429	56/12	672	15/3	4	-	
13	c	149	55/23	191	23/12	672	15/3	4

order 1 of substitution is used. Component n always contains the original target list and the order of execution was component 1,2,...,n. The sort costs are not included. One can see from this table that usually the most reduction in size from the sort is gained after the first component. However since the relation monotonically decreases in size, the sorting cost of the subsequent components also decreases.

Note that for queries with structures (a) or (b), the dominant component in terms of cost is usually the last component. For query 5, this is not true but the size of the variable which appeared in the target list with the joining variable was approximately 1/10th of the largest variable. Thus, one would expect the cost of the last component to be small. For queries with structure (c), one would expect the last component to be the smallest also. This component contains two joining variables which have already been reduced in size. So hypothesis (2) appears to be true for queries of structures (a) or (b) when the size of the remaining target list variable is not very small compared to the other relations. However it is felt that this observation is not really that helpful since it really does not give a good indication of the total cost of processing. Also, the last component may have the largest cost but there could be others which are quite close.

8.3.2 Reduction vs. Tuple Substitution

In Chapter 5, reduction was proposed as a processing technique and an analysis was performed in an attempt to determine exactly when reduction should be used prior to tuple substitution. The results of that analysis can be summarized as follows:

1. When the variable selected is a joining variable, substitution for that variable will require less total page accesses than reduction on that variable followed by substitution for it first in each component.
2. When the variable selected (X_i) is not in the target list but is involved in a two-variable clause with a variable in the target list, substitution for the selected variable will be better than reduction assuming that X_i is selected for substitution first in the component in which it appears.
3. When the variable selected (X_i) is in the target list and appears only in a clause involving three or more variables or not at all in the qualification, and, if substitution results in a connected subquery, reduction will be less expensive than substitution for X_i when X_i is substituted first in the reduction component containing the target list.
4. When the variable selected (X_i) is not in the target

list and does not appear in any two-variable clauses with a variable in the target list and substitution for X_i results in a subquery whose graph is explicitly disconnected, reduction does better than substitution for X_i when X_i is substituted first in the first reduction component.

In all other cases, no definite conclusion could be stated but it was felt that reduction would perform better in general. The assumptions made for the purpose of the analysis were that none of the relations had a keyed structure, no sort operation to remove duplicates was performed between intermediate reduction components, and that the same variable would be selected for substitution after a reduction as before.

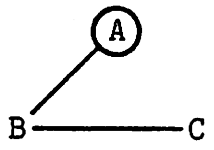
Obviously the results of this analysis depend upon which variable is selected for substitution. In order to test all cases, measurements were taken for each variable that would be selected using any of the criteria discussed in Section 8.2. Also, since is was not known if the assumptions made for the analysis were valid, experiments were performed where the assumptions were relaxed.

The analysis of Chapter 5 used the guidelines presented there for how far a query should be reduced given the role of the variable selected for substitution. Since the experiments can fully reveal the effect of only the first decision, all levels of reduction were run to get an idea of the total processing costs.

The purpose of the measurements in this section is, first, to evaluate the results of the theoretical analysis of Chapter 5. Second, it is felt that the assumption that the same variable will be selected before and after reduction is too strict. So, this assumption will be relaxed and the results presented. Third, the hypothesis that storage structure has no effect on the decision as to whether to use reduction or substitution will be tested.

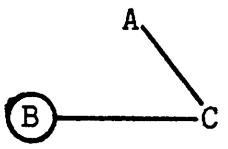
The following graphs depict the structure and characteristics of the queries used in the experiments.

Query 1



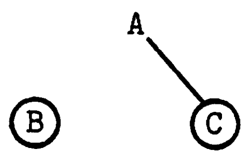
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 51 \quad P_C = 14$

Query 2



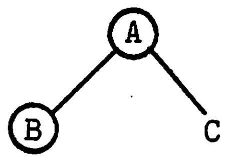
$t_A = 66 \quad P_A = 15$
$t_B = 56 \quad P_B = 13$
$t_C = 150 \quad P_C = 39$

Query 3



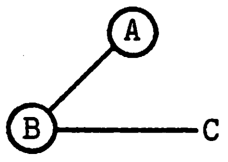
$t_A = 66$ $P_A = 15$
$t_B = 56$ $P_B = 13$
$t_C = 150$ $P_C = 39$

Query 4



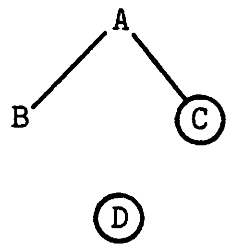
$t_A = 62$ $P_A = 32$
$t_B = 62$ $P_B = 17$
$t_C = 56$ $P_C = 29$

Query 5



$t_A = 66$ $P_A = 15$
$t_B = 51$ $P_B = 14$
$t_C = 56$ $P_C = 13$

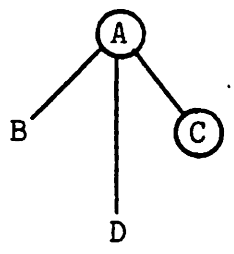
Query 6



$t_A = 98$ $P_A = 50$
$t_B = 79$ $P_B = 17$

$t_C = 62$ $P_C = 17$
$t_D = 40$ $P_D = 3$

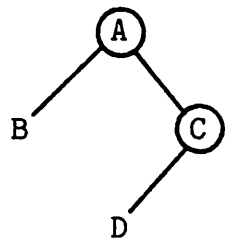
Query 7



$t_A = 56$ $P_A = 29$
$t_B = 62$ $P_B = 17$

$t_C = 112$ $P_C = 57$
$t_D = 97$ $P_D = 26$

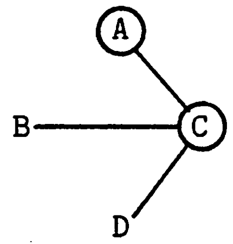
Query 8



$t_A = 62$ $P_A = 32$
$t_B = 51$ $P_B = 14$

$t_C = 56$ $P_C = 29$
$t_D = 62$ $P_D = 17$

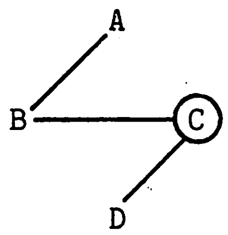
Query 9



$t_A = 62$ $P_A = 32$
$t_B = 62$ $P_B = 17$

$t_C = 138$ $P_C = 70$
$t_D = 51$ $P_D = 14$

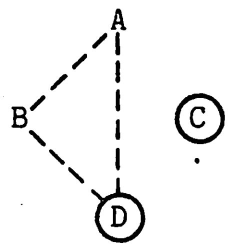
Query 10



$t_A = 62$ $P_A = 17$
$t_B = 51$ $P_B = 14$

$t_C = 62$ $P_C = 32$
$t_D = 56$ $P_D = 29$

Query 11



$t_A = 51$ $P_A = 14$
$t_B = 56$ $P_B = 29$

$t_C = 62$ $P_C = 17$
$t_D = 62$ $P_D = 32$

The first measurements to be presented are those concerning the hypotheses of Chapter 5. The results will be broken down into categories. Table 8.14 presents the measurements when a variable selected for substitution is a joining variable (hypothesis 1). The second table contains the results when the variable selected is in the target list (hypothesis 3), and the third is when the variable selected is not in the target list (hypotheses 2 & 4). Note that there will be some queries in the second and third tables which do not fit the criterion of the hypotheses, thus which hypothesis is applicable, if any, will be indicated.

Each table will include the query number, the variable selected for substitution, the cost if that variable is substituted, and the cost if that query is reduced using the guidelines for how much to reduce presented in Chapter 5. Since there are various orders of substitution possible even after the first variable is selected, the cost included will be the minimum one. The reduction cost is the cost assuming the selected variable is substituted first in whichever components it may appear. Also included are each of these costs when keyed structures are used.

TABLE 8.14. Substitution variable is a joining variable.

Query No.	Var	Substitution Cost non-keyed	Substitution Cost keyed	Reduction Cost non-keyed	Reduction Cost keyed
1	B	793	255	794	257
2	C	2343	479	2346	482
3	C	2261	636	2351	726
4	A	1083	157	1169	210
5	B	1015	469	1120	485
6	A	3015	1637	3176	1798
6	C	4516	1783	3399	713
7	A	3670	3357	3698	3386
8	A	4257	2230	4257	2230
8	C	11266	6048	11349	6182
9	C	4329	3145	1677	261
10	B	3367	1951	3414	2013
10	C	927	357	942	372
11	D	14751	14751	2091	762

From Table 8.14, it can be seen that generally when the variable selected for substitution is a joining variable, substitution for that variable is better than reduction followed by substitution for it first in each component. However, there are exceptions even to this. But note that the existence of keyed structures does not affect the relative costs.

Hypothesis 3 proposes that reduction does better than substitution under its conditions. From Table 8.15, it can be seen that whether the hypothesis is true or not, reduction should be the technique selected to achieve a minimum processing cost. This result makes sense because generally when a variable appears in the target list and is not a joining variable, it will not be very dominant in the qualification. Thus substituting for it gains essentially nothing and reduction can gain quite a lot. Again, the storage structure did not change the relative costs.

The results of Table 8.16 indicate that reduction does better than substitution in almost all cases where the substitution variable is not in the target list. The main reason that reduction loses in query 4 is that the result of the first reduction component contains $|R(A)||R(C)|$ tuples, since all tuples are retrieved for each substituted value. If either the operation to remove duplicates is included or the order of substitution for the first component is changed, the reduction cost drops dramatically. However, these results refute hypothesis 2 and verify hypothesis 4.

Examining hypothesis 2 and the queries where it is applicable, there is a possible explanation for why it is false. Since X_1 is not in the target list, it does not appear in the last component, only in the first. Thus, reduction is free to select any variable for substitution in all components except the first. This flexibility allows for more optimization. Also, in Conjecture 2, the cost comparisons are made using tuples only. The joining variable will likely have a larger reduction in pages than in tuples since the tuple width will be restricted. These two factors could easily outweigh the advantage that suggested Conjecture 2.

TABLE 8.15. Substitution variable is in the target list.

Query No.	Var	Hypo. 3 Applicable	Substitution Cost non-keyed	Substitution Cost keyed	Reduction Cost non-keyed	Reduction Cost keyed
1	A	no	980	892	744	208
2	B	no	2265	1450	1888	263
3	B	yes	144760	12713	1944	319
4	B	no	1986	540	364	364
5	A	no	2458	2054	578	323
6	D	yes	128083	51420	2508	1130
9	A	no	113296	88470	2114	698
11	C	yes	112716	34038	1914	645

TABLE 8.16. Substitution variable is not in the target list.

Query No.	Var	Applicable Hypothesis	Substitution Cost non-keyed	Substitution Cost keyed	Reduction Cost non-keyed	Reduction Cost keyed
1	C	-	3942	2526	820	316
2	A	-	9840	7530	2708	351
3	A	2	7970	5660	4052	1695
4	C	2	13496	13496	58625	4921
5	C	2	3260	2732	1753	1189
7	B	2	168516	154143	5034	3372
8	B	2	28681	4483	5042	1110
9	D	2	~72569991	-	~209270	-
10	A	-	1335	633	902	198
10	D	2	~43486	-	8780	3291
11	A	4	15778	11086	1597	1570
11	B	4	15829	14424	3074	1769

From the results in the last three tables, and under the assumptions made in Chapter 5, substitution is the preferred method only when the variable selected is a joining variable. In all other cases, reduction usually performs better and there are even cases when the variable selected is a joining variable and reduction is better. Due to these conclusions, it appears that the analysis performed in Chapter 5 forces too many assumptions on the environment to be practically applicable.

This result enforces even more the desire to examine reduction when certain of the assumptions are relaxed, specifically the assumption concerning the continuity of the selection criterion after reduction. The evaluation of queries which reduce to all two-variable components provide a hint to the fact that the role of a variable before and after reduction can change. And, the results of the variable selection criteria evaluation show that the structure of a query plays a dominant role in selecting a minimum processing path. So these factors should be considered for the processing of each component individually.

The following table (Table 8.17) thus presents for each query, the variable selected for substitution on the basis of the query structure criterion as stated in Section 8.2.2. Then, the cost of substituting for that variable is compared to the cost of reduction, the selection criteria of Section

8.2 being applied to each component individually. The resulting sequence of components is also included. Finally, the minimum attainable reduction cost and the corresponding sequence of components are given. The costs for both keyed and non-keyed structures are included.

The substitution costs using the query structure criterion are generally the minimum substitution cost. In the cases where it´s not, there is little difference between it and the minimum. Thus the results of this table show that reduction on the variable selected results in a lower processing cost than substitution for that variable. These reductions were performed without any operation to remove duplicates. If this is included, the reduction costs should decrease because the intermediate result ranges are bounded by the size of the joining variable´s range, so the sort costs should be small generally. In the cases where the reduction cost for the selected variable and the minimum reduction cost do not match, the difference is usually due to the order in which the components were executed. So the heuristic of performing the component involving the relation with the smallest number of pages first may not be a good one. For query 10, where the minimum reduction cost corresponds to reducing on a different variable, notice from its graph that variable B is also a plausible selection using the query structure criterion.

TABLE 8.17. Reduction vs. substitution.

Query No.	Var	Substitution non-keyed	Substitution keyed	Reduction non-keyed	Reduction keyed	Reduction components	Minimum Reduction non-keyed	Minimum Reduction keyed	Minimum Reduction components
1	B	806	257	744	208	B,C;A,B	744	208	B,C;A,B
2	C	2343	479	1888	263	C,A;B,C	1888	263	C,A;B,C
3	C	2261	636	1944	319	C,A;B,C	1944	319	C,A;B,C
4	A	1083	157	354	354	A,C;B,A	354	354	A,C;B,A
5	B	10154	469	578	323	B,C;A,B	578	323	B,C;A,B
6	A	3015	1637	519	519	A,B;C,D,A	519	519	A,B;C,D,A
7	A	3670	3357	957	643	A,B;A,D;C,A	953	641	A,D;A,B;C,A
8	C	11266	6048	1302	471	C,D;A,C,B	1159	302	C,B;C,D;C,A
9	C	6249	4920	4458	3042	C,D;C,B;C,A	1677	261	C,B;C,D;C,A
10	C	927	357	655	419	C,D;B,C,A	441	159	B,A;C,B,D
11	D	14751	14751	1914	645	D,A,B;C,D	1914	645	D,A,B;C,D
12	B	19815	19815	1992	1128	B,D;A,B,C	1992	1128	B,D;A,B,C

We can summarize these conclusions as follows: Whenever a query is reducible and the query structure criterion is applicable, the query should be reduced using the guidelines of Chapter 5 to determine how far to reduce it. Then each component should be evaluated by applying the variable selection criteria to it individually. In general, when an intermediate component involves only two variables, the variable in the target list, which is the joining variable, should be selected for substitution. If this policy is used even for three or more variable components, the cost of performing a sort to remove duplicate tuples will be small and usually affordable. Note that if the option of removing duplicates prior to substitution as proposed in Chapter 6 is used, the sort operation between intermediate components will generally not be needed.

This conclusion is for the general case. Obviously, one can generate queries where reduction will lose due to the distribution of the data. However the analysis came up with only two cases where substitution would definitely win. The first one, hypothesis 2, was disputed by the measurements. The second one, when the substitution variable is a joining variable, was supported by the measurements under the assumptions of the analysis. But when the continuity of a selection criterion assumption was relaxed, this result was no longer valid. The relaxation of this assumption allows reduction more flexibility to take advantage of the effect of one component on the next. Thus in a query such as

```
    RETRIEVE (X.a, Y.a)
        WHERE  X.b = Z.b
```

the order: 1 - X,Z, 2 - Y,X could result in a lower cost due to the effect of the first component. Substitution cannot take advantage of this while reduction can. So even though the measurements are for a limited number of queries and data, the reasoning behind their results is sound and the results can therefore be stated as general conclusions.

In Figure 8.2 a graphical representation of the results of this section is presented. This figure contains a plot of $\frac{\text{cost}}{P_{avg}}$ vs. no. of variables on semi-log paper. Included in this graph is the minimum measured processing cost (●), in data pages accessed, for the test queries (assuming non-keyed structures) and certain bounds on these processing costs. Obviously, for any query with any number of variables, ΣP_i (□) is a lower bound. There are two upper bounds which were compared. Since the test queries did not all reference the same relations, these upper bounds appear on the graph as a range of values. The first is $\prod t_i$ (▲). Clearly, this is a greatest upper bound and this bound will generally not be attained due to the fact that OVQP accesses pages. The graph supports this observation. The second upper bound considered is $\prod P_i$ $(\underline{\triangle})$. Because of preprocessing of one-variable restrictions and reduction, this can be used as an upper bound.

If we examine Figure 8.2, it can be seen that for two-variable queries, $\prod P_i$ is actually a lower bound rather than an upper bound. This is to be expected in this case because reduction is not an available processing alternative. The only queries whose measured processing costs fall below $\prod P_i$ are those with a single target-list variable.

However, whenever the number of variables is more than two, the actual cost falls below $\prod P_i$ and a large distance below $\prod t_i$. In fact, the measured costs are generally much closer to the lower bound of $\sum P_i$ than to either of the upper bounds. Since the minimum processing costs usually correspond to some reduction for the queries involving three or more variables, this attests to the fact that the use of reduction leads to more efficient processing.

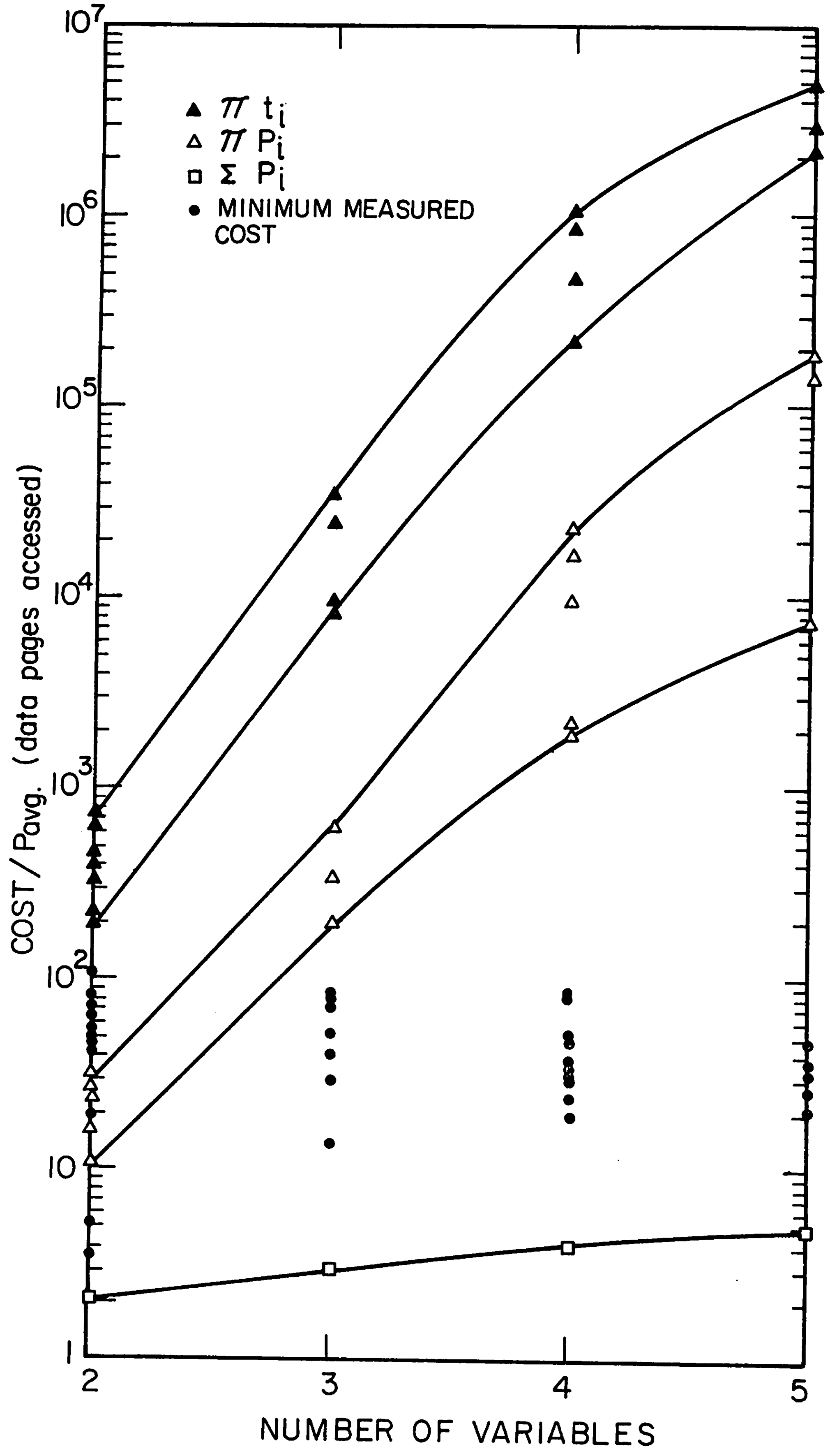


FIGURE 8.2. Cost/(avg. no. of pages) vs no. of variables.

CHAPTER 9

CONCLUSIONS

In this dissertation we have examined the topic of efficient processing of queries in a relational database management system. Techniques which can be used at different stages of the processing and which depend on various characteristics of the query environment have been proposed in an attempt to achieve a minimum processing cost for a query. In this chapter, the highlights of this work will be briefly summarized and directions for future research will be indicated.

9.1 Query Transformations

In Chapter 4, queries were examined to determine what characteristics a query should ideally possess, independent of the data being referenced, to lead to more efficient processing. Then using these characteristics as goals, a set of transformations which can be applied to any query to acquire these characteristics were stated. The idea behind this technique is similar to that used in compiler optimization. Some work on developing and testing these techniques has been done in a relational algebra environment [HALL75, SMIT75]. However, no work has been done specifically to evaluate its usefulness in a relational-calculus based environment.

9.2 Query Processing

In Chapter 5, the concept of tuple substitution, or nested iteration, was defined and shown to be a sufficient tool for processing any query. However this tactic is equivalent to creating the entire cross product specified by the query on a tuple-by-tuple basis. A second concept called reduction, which exploits the structural characteristics of the query, was then introduced. Since reduction can result at best in only a sequence of two-variable subqueries, it is necessary to use a combination of reduction and tuple substitution to process a query to completion. From the theoretical analysis of Chapter 5, it appeared that there were only a limited number of cases where reduction should be applied to a query prior to substitution. However several assumptions were forced on the query environment in order to perform this analysis, and it turned out that some of these assumptions were too restrictive. The measurements performed for Chapter 8 to compare the usage of reduction and substitution have shown that reduction should always be applied if possible. This result is true when each component resulting from the reduction is examined as an individual query and techniques for efficient processing applied to it without any assumptions related to decisions made for the original query.

All of the ideas discussed thus far are independent of the specific data and its structure that the query references. Certain steps which could be taken to tailor the processing algorithm to this information were detailed in Chapter 6. These steps included preprocessing of one-variable restrictive clauses, performing a projection to reduce the number of tuples prior to tuple substituting for a relation, and dynamically modifying the storage structure of a relation to allow more efficient access. Obviously the degree of effectiveness of these options depends on the value distribution of the domains involved. However both the analysis in Chapter 6 and the measurements in Chapter 8 have shown that knowledge of the distributions would have little effect on the basic decision as to whether these options should be exercised.

However in the case of dynamic storage modification, it was shown that this information would be useful in deciding between modifying the primary structure or building a secondary index. Whenever the expected number of tuples retrieved per substitution is one or less or whenever there is a high percentage of substitution values for which no tuples will qualify, the decision should be made in favor of indexing. If statistics are not available to make this choice and the wrong decision is made, the results can be catastrophic.

9.3 Variable Selection

Selecting the variable whose range is to be tuple substituted is clearly a critical step in the processing of any query. There are several factors which should influence this decision. These include the size in tuples and pages of each relation involved, the storage characteristics of those relations, the structure of the query and the distribution of the domains referenced by the query. All information except the distributions is readily available and a reasonable selection strategy using those factors was developed. For two-variable queries, the variable which minimizes the ratio $\frac{t}{P_{eff}+1}$ should be selected for substitution. P_{eff} is the number of data pages to be accessed considering the effect of the query structure and storage structure.

For queries involving three or more variables there is a more specific ordering:

1. query structure, specifically the number of two variable clauses in which a variable appears,
2. size, specifically the ratio tuples/(pages + 1),
3. in the case of a close decision on the basis of another criterion, storage structure characteristics should be used as the determining factor.

If information depicting the distribution of the domains is available to make this decision, it is necessary to develop a cost estimate function to consider the effect of all of these factors. This cost function must be reasonably accurate but it is also necessary to limit it in a certain way. For an n-variable query, there are n! possible orderings for tuple substitution and perhaps more than one available reduction. To estimate and compare all of these when n is large would be quite expensive. Thus some way of limiting the number of candidate paths must be developed.

In conclusion, this work has presented several techniques which will lead to more efficient processing of queries in a relational database environment. As such, it is an attempt to demonstrate that a database system based on the concept of relations is a plausible organization. The main area for future research is in the usage and maintenance of statistical information to reflect the database environment. This includes determining what information would be valuable, and efficient means for gathering and maintaining it. Since the areas discussed here are not the only ones which could benefit from such information, developments in this direction would be advantageous to the entire database system.

APPENDIX A

RELATIONAL ALGEBRA OPERATORS

Let R be a relation and $D_1, D_2, \cdots, D_n$ the domains of R. Then R is a subset of the cartesian product of its domains, $R = D_1 x D_2 x \cdots x D_n$.

Project: a unary operator which acts on a relation R by eliminating some of its columns and then deleting any duplicate tuples which might result in the subset of original columns. Let $\hat{i} = (i_1, i_2, \cdots, i_m)$ be a subset of the integers 1 through n. Then the projection of R on $\{D_j, j \in \hat{i}\}$, denoted $R[D_{\hat{i}}]$, is defined by

$$R[D_{\hat{i}}] = \{(r_{i_1}, r_{i_2}, \cdots, r_{i_m}): (r_1, r_2, \cdots, r_n) \in R\}$$

Let R have domains $D_1, \cdots, D_n$ and S have domains $D'_1, \cdots, D'_m$. If $r \in R$ and $s \in S$, we shall denote by (r,s) the concatenation of r and s, i.e.,

$$((r_1, \cdots, r_n), (s_1, \cdots, s_m)) = (r_1, \cdots, r_n, s_1, \cdots, s_m)$$

The expanded cartesian product R x S is defined by

$$R \times S = \{(r,s): r \in R \text{ and } s \in S\}$$

A distinction can be made between the cartesian product R x S which is a collection of pairs r and s and R x S which is a collection of the concatenations of r and s, but it is hardly worth making.

Restrict or Select: Let $P(\overline{x})$ be a truth function defined on the cartesian product $D_1 x D_2 x \cdots x D_n$. That is, for each $\overline{x}$ in $D_1 x D_2 x \cdots x D_n$, $P(\overline{x})$ has a unique value which is either 1 (true) or 0 (false). Then the subset $R[P]$ of R for which P is true is called the restriction of R relative to P.

$$R[P] = \{r: r \in R \text{ and } P(r) = 1\}$$

Let $\ddot{x}$ stand for any of the comparison operators: $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, and let the predicates $\neg P$ (not P), $P \wedge Q$ (P and Q), P v Q (P or Q), and $P \Longrightarrow Q$ (P implies Q) be defined as in standard logic.

Then let θ_0 be the set of predicates on $\prod D_i$ defined by the properties:

(a) $D_i \ddot{x}\, k \in \theta_0$ for each i and every constant k in D_i.

(b) $D_i \ddot{x}\, D_j \in \theta_0$ for $1 \leq i \leq j \leq n$.

(c) $P \in \theta_0 \Longrightarrow \neg P \in \theta_0$.

(d) $P, Q \in \theta_0 \Longrightarrow P \wedge Q \in \theta_0$ and $P \vee Q \in \theta_0$.

(e) θ_0 is minimal with respect to (a) through (d).

Then for each $P \in \theta_0$, the restriction $R[P]$ is well-defined.

Join: Let R and S be two relations with domains $D_1, \cdots, D_n$ and $D_{n+1}, \cdots, D_{n+m}$ respectively. Let P be a predicate on

$D_1, \ldots, D_{n+m}$ belonging to $\mathfrak{H}_0$. Then the join of R with S relative to P is defined by

$$R[P]S = (R \times S)[P]$$

If the condition P in the join is an equality between two domains, one from each of the two relations participating in the join, then the join is said to be an equi-join. An equi-join clearly always has two identical columns, and if one of them is eliminated by projection, then the result is known as the natural join.

APPENDIX B

ANALYSIS OF MODIFYING TO HASH STRUCTURE vs. SORTED STRUCTURE

This analysis attempts to model these modify operations according to the way they are performed in the current implementation of INGRES. It is felt that the cost associated with reading and writing the pages of the relation being modified will, in general, be larger than the cost of computing the hash function or comparing tuple values. For this reason, the costs will be estimated assuming that the I/O is the dominant factor. That is, the costs of computing the hash function or comparing tuple values will be ignored.

There are certain system defined constants and some notation which will be used throughout the discussion which will be explained first. A page is the basic unit of access between memory and secondary storage devices. For UNIX, a page is 512 bytes (UPAGE). A page storing tuples of a relation also contains certain system information so that all 512 bytes are not available for data. Currently, IPAGE = 498 bytes are useable for data within INGRES. However, tuples cannot be split between pages. So, if a tuple contains 250 bytes, only a single tuple will fit on one page. This is true for all pages containing data from relations in INGRES. However, the sorting routine uses UNIX size pages and does not force a page to contain only complete tuples.

The sorting routine reads as many tuples as possible into a core buffer and then sorts them. The size of this core buffer (BUF) determines the size and number of the intermediate merge files required. Currently, max(BUF) = 63 pages.

It will be assumed in the following discussion that a relation R is being modified. t is the number of tuples in R, w is the width (in bytes) of a tuple of R, P is the number of INGRES pages in secondary storage that R occupies, and S is the number of UNIX pages that R occupies.

First, for hashing, all original pages of R must be read. Then, for each tuple, the hash address is computed and the tuple written to the appropriate output page. If the tuples are input in a random order, it cannot be assumed that two consecutive tuples will hash to the same output page. So, for each tuple, the output page on which the last tuple was inserted must be written and the page on which the current tuple belongs must be read. This results in a write and a read of the output file for each tuple in R. Even if a duplicate tuple is encountered, the page it hashes to must be read to determine that it is a duplicate. This results in a cost for hashing of P + 2t. However, this function does not include the cost of examining overflow pages which might occur in the resulting relation. The following table from [SEVE74] lists the expected number of overflow tuples as a percentage of the number of stored tuples using a chained technique in a separate overflow area. The only load factor included is 1 since the costs are being considered for modifying temporary relations.

bucket size	load factor 1.0
1	36.79
2	27.07
3	22.40
4	19.54
5	17.55
6	16.06
7	14.90
8	13.96
9	13.18
10	12.51
12	11.44
14	10.60
16	9.92
18	9.36
20	8.88
25	7.95
30	7.26
35	6.73
40	6.29
45	5.94
50	5.63
60	5.14
70	4.76
80	4.46
90	4.20
100	3.99

TABLE A.1. Overflow Records per Record Stored (percent).

Since this percentage depends upon the bucket size (no. of tuples per page), for accurate calculations the bucket size must be determined and the appropriate percentage used. For purposes of comparison here, an average bucket size of 20 will be used. Thus, approximately 9% of the tuples will reside on overflow pages and an additional read will be required for those tuples. This results in a final cost for hashing of:

$$C(hash) = P + 2.888t$$

Note that the percentages presented in Table A.1 assume a random distribution for the key domain.

For sorting, the first thing done is that the original relation R is copied to a UNIX file (with UPAGE bytes per page). This file is then read in blocks of BUF pages. Each block of BUF pages is sorted and written to an intermediate merge file. Then a 7-way merge is performed as many times as necessary. If F is defined as the number of original intermediate merge files, then no matter what F is, each page must be read and written at least once. If F=1, then once is enough to complete the sort. Otherwise additional reads as defined by the following table must be done.

F		f
2-7	$(7^0+1 - 7^1)$	1
8-49	$(7^1+1 - 7^2)$	2
50-343	$(7^2+1 - 7^3)$	3
344-2401	$(7^3+1 - 7^4)$	4
⋮	⋮	⋮

Thus, we can define

$$f(x) = \begin{matrix} 0 & \text{if } x \leq 1 \\ \log_7 x & \text{if } x \geq 1 \end{matrix}$$

Finally, once the sort/merge is complete, the UNIX file is copied back into the relation R and the directories are built to complete the ISAM structure. The cost associated with building the directories is approximately 20% of the number of pages for the first level, 20% of that number for the second level, etc. The cost used will be 25% of the number of pages. It should be noted that if this method is used strictly to remove duplicates and the tuples will not be accessed using the ISAM structure, the process of building the directories can be eliminated.

This results in a cost for ISAM of

$$C(ISAM) = P + S + 2S|f(\frac{S}{BUF}) + 1| + S + 2.5P$$

Note that since $P = \frac{t}{int(\frac{IPAGE}{w+2})}$ and $S = \frac{tw}{UPAGE}$ the costs of modifying to ISAM and hash can be compared as a function of t and w. The following table presents the results of such comparisons using IPAGE = 498, UPAGE = 512, BUF = 63, and the percentages of Table A.1.

The letter in each box indicates the operation with the least cost for the associated (t, w) pair.

It can be seen that whenever the tuple width is greater than 164 bytes, hashing is the less expensive operation regardless of the cardinality of the relation. Notice that a width of 165 bytes corresponds to two tuples per page while 164 corresponds to 3 tuples per page. In actuality, this boundary between hashing and ISAM is determined by the page capacity for a given relation.

However since the cases where comparison is of importance are all dealing with temporary relations whose domains are only the subset of the original domains referenced by the query, usually the tuple width will be small. If this is true, modifying to an ISAM structure will be the least expensive of the two operations.

tuples	width								
	<50	50	55	60	75	90	105	125	>165
10^2	I	I	I	I	I	I	I	I	H
10^3	I	I	I	I	I	I	I	H	H
10^4	I	I	I	I	I	I	H	H	H
10^5	I	I	I	I	I	H	H	H	H
10^6	I	I	I	I	H	H	H	H	H
10^7	I	I	I	H	H	H	H	H	H
10^8	I	I	H	H	H	H	H	H	H
10^9	I	H	H	H	H	H	H	H	H

TABLE A.2. ISAM vs HASH cost as a function of number of tuples and tuple width.

REFERENCES

ALLE72 Allen, F.E. and Cocke, J., "A Catalogue of Optimizing Transformations", DESIGN AND OPTIMIZATION OF COMPILERS, R.Rustin (ed.), Prentice-Hall, 1972.

ALLM76 Allman, E., Stonebraker, M. and Held, G., "Embedding a Relational Data Sublanguage in a General Purpose Programming Language", Univ. of California, Berkeley, ERL Memo No. ERL-M564, October 1976.

ASTR75 Astrahan, M.M., Lorie, R.A., "SEQUEL-XRM: A Relational System", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.

ASTR75a Astrahan, M.M., Chamberlin, D.D., "Implementation of a Structured English Query Language", Comm. ACM, Vol. 18, No. 10, October 1975.

ASTR76 Astrahan, M.M. et al, "System R: A Relational Approach to Data Base Management", ACM Trans. on Database Systems, Vol. 1, No. 2, pp. 97-137, June 1976.

BACH74 Bachman, C.W., "The Data Structure Set Model", Proc. 1974 ACM-SIGMOD Debate "Data Models: Data Structure Set versus Relational" (ed. R. Rustin), Ann Arbor, Michigan, May 1-3, 1974.

BAUE74 Bauer, F.L. and Eickel, J. (eds.), COMPILER CONSTRUCTION: AN ADVANCED COURSE, Springer-Verlag, New

York, 1974.

BJOR73 Bjorner, D., Codd, E.F., Deckert, K.L., Traiger, I.L., "The Gamma Zero n-ary Relational Data Base Interface: Specifications of Objects and Operations", IBM San Jose Research Report RJ1200, April 11, 1973.

BLAS75 Blasgen, M.W. and Eswaran, K.P., "On the Evaluation of Queries in a Relational Data Base System", IBM San Jose Research Report RJ-1745, April 1976.

BOYC74 Boyce, R.F., Chamberlin, D.D., King, W.F., III, Hammer, M.M., "Specifying Queries as Relational Expressions: SQUARE", Data Base Management (Proc. IFIP Working Conference, Corsica, April 1-5, 1974), pp. 169-177, North-Holland, Amsterdam, 1974.

CHAM74 Chamberlin, D.D. and Boyce, R.F., "SEQUEL: A Structured English Query Language", Proc. ACM-SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May 1-3, 1974.

CHAM76 Chamberlin, D.D., "Relational Data-Base Management Systems", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 43-66.

CODA71 Committee on Data Systems Languages, "CODASYL Data Base Task Group Report", ACM, New York, April 1971.

CODD70 Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

CODD71	Codd, E.F., "A Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Calif., November 11-12, 1971.
CODD71b	Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Calif., November 11-12, 1971.
CODD71c	Codd, E.F., "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia 6, DATA BASE SYSTEMS, pp. 65-98, Prentice-Hall, New York, 1971.
CODD74	Codd, E.F. and Date, C.J., "Interactive Support for Non-Programmers: The Relational and Network Approaches", Proc. 1974 ACM-SIGMOD Debate "Data Models: Data Structure Set versus Relational" (ed. R. Rustin), Ann Arbor, Michigan, May 1-3, 1974.
CZAR75	Czarnik, B., Schuster, S. and Tsichritzis, D., "ZETA: A Relational Data Base Management System", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.
DANT63	Dantzig, G.B., LINEAR PROGRAMMING AND EXTENSIONS, Princeton University Press, Princeton, N.J., 1963.
DATE72	Date, C.J., "Relational Data Base Systems: A Tutorial", Proc. Fourth International Symposium on Computer and Information Sciences (COINS IV), Miami

Beach, Florida, December 14-16, 1972, Plenum Press.

DATE74 Date, C.J. and Codd, E.F., "The Relational and Network Approaches: Comparison of the Application Programming Interfaces", Proc. 1974 ACM-SIGMOD Debate "Data Models: Data Structure Set versus Relational" (ed. R. Rustin), Ann Arbor, Michigan, May 1-3, 1974.

DATE75 Date, C.J., AN INTRODUCTION TO DATA BASE SYSTEMS, Addison-Wesley, 1975.

FARL75 Farley, J.H.G. and Schuster, S.A., "Query Execution and Index Selection for Relational Data Bases", Technical Report CSRG-53, Computer Systems Research Group, Univ. of Toronto, Toronto, Canada, March 1975.

GOLD70 Goldstein, R.C. and Strnad, A.L., "The MACAIMS Data Management System", Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access, Houston, Texas, November 15-16, 1970.

GOTL75 Gotlieb, L.R., "Computing Joins of Relations", Proc. ACM-SIGMOD Conference, San Jose, Calif., May 14-16, 1975.

HALL74 Hall, P.A.V. and Todd, S.J.P., "Factorisations of Algebraic Expressions", IBM Scientific Centre Report UKSC 0055, Peterlee, England, April 1974.

HALL74a Hall, P.A.V., "Common Sub-expression Identification in General Algebraic Systems", IBM Scientific

Centre Report UKSC 0060, Peterlee, England, November 1974.

HALL75 Hall, P.A.V., "Optimisation of a Single Relational Expression in a Relational Data Base System", IBM Scientific Centre Report UKSC 0076, Peterlee, England, July 1975.

HALL75a Hall, P.A.V., Todd, S.J.P. and Hitchcock, P., "An Algebra of Relations for Machine Computation", IBM Scientific Centre Report UKSC 0066, Peterlee, England, January 1975.

HELD75 Held, G. and Stonebraker, M., "Networks, Hierarchies, and Relations in Data Base Management Systems", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.

HELD75a Held, G., Stonebraker, M. and Wong, E., "INGRES: A Relational Data Base System", Proc. National Computer Conference, Anaheim, Calif., May 19-22, 1975.

HELD75b Held, G. and Stonebraker, M., "Storage Structures and Access Methods in the Relational Data Base Management System INGRES", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.

HELD75c Held, G.D., "Storage Structures for Relational Data Base Management Systems", Ph.D. Dissertation, Univ. of California, Berkeley, ERL Memo No. ERL-M533, 1975.

IBM70 IBM Corp., "IMS/360 Applications Description

Manual", IBM, White Plains, New York, GH-20-0765.

JOHN74 Johnson, S.C., "YACC, Yet Another Compiler-Compiler", UNIX Programmer´s Manual, Bell Telephone Labs, Murray Hill, N.J., July 1974.

KNUT73 Knuth, D., THE ART OF COMPUTER PROGRAMMING, Vol. 3, Addison-Wesley, Reading, Mass., 1973.

LORI74 Lorie, R.A., "XRM - An Extended (n-ary) Relational Memory", IBM Scientific Centre Report G320-2096, Cambridge, Mass., January 1974.

LUM70 Lum, V.Y., "Multiattribute Retrieval with Combined Indices", Comm. ACM, Vol. 13, No. 11, pp. 660-665, November 1970.

LUM71 Lum, V.Y., Yuen, P.S.T. and Dodd, M., "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files", Comm. ACM, Vol. 14, No. 4, pp. 228-239, April 1971.

MCDO75 McDonald, N. and Stonebraker, M., "CUPID: The Friendly Query Language", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.

MULL71 Mullin, J., "Retrieval-Update Speed Tradeoffs Using Combined Indices", Comm. ACM, Vol. 14, No. 12, pp. 775-776, December 1971.

MYLO75 Mylopoulos, J., Schuster, S.A. and Tsichritzis, D., "A Multi-Level Relational System", Proc. National Computer Conference, Anaheim, Calif., May 19-22, 1975.

NOTL72 Notley, M.G., "The Peterlee IS/1 System", IBM Scientific Centre Report UKSC-0018, March 1972.

PALE72 Palermo, F.P., "A Data Base Search Problem", Fourth International Symposium on Computer and Information Science (COINS IV), Miami Beach, Florida, December 1972, Plenum Press.

PECH75 Pecherer, R.M., "Efficient Retrieval in Relational Data Base Systems", Ph.D. Dissertation, Univ. of California, Berkeley, ERL Memo No. ERL-M547, 1975.

PECH75a Pecherer, R.M., "Efficient Evaluation of Expressions in a Relational Algebra", Proc. ACM Pacific Conference, San Francisco, April 17-18, 1975.

PECH76 Pecherer, R.M., "Efficient Exploration of Product Spaces", Proc. ACM-SIGMOD Conference, Washington, D.C., June 2-4, 1976.

RITC74 Ritchie, D. and Thompson, K., "The UNIX Time Sharing System", Comm. ACM, Vol. 17, No. 7, pp. 365-375, July 1974.

RITC74a Ritchie, D.M., "C Reference Manual", UNIX Programmer´s Manual, Bell Telephone Labs, Murray Hill, N.J., July 1974.

ROTH72 Rothnie, J.B., "The Design of Generalized Data Management Systems", Ph.D. Dissertation, Dept. of Civil Engineering, MIT, September 1972.

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

Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May 1-3, 1974.

ROTH75 Rothnie, J.B., "Evaluating Inter-Entry Retrieval Expressions in a Relational Data Base Management System", Proc. National Computer Conference, Anaheim, Calif., May 19-22, 1975.

SCHM75 Schmid, H.A. and Bernstein, P.A., "A Multi-Level Architecture for Relational Data Base Systems", Proc. International Conference on Very Large Data Bases, Framingham, Mass., September 22-24, 1975.

SEVE74 Severance, D.G., "Identifier Search Mechanisms: A Survey and Generalized Model", ACM Computing Surveys, Vol. 6, No. 3, pp. 175-194, September 1974.

SIBL74 Sibley, E.H., "On the Equivalences of Data Based Systems", Proc. ACM-SIGMOD Debate "Data Models: Data Structure Set versus Relational" (ed. R. Rustin), Ann Arbor, Michigan, May 1-3, 1974.

SMIT75 Smith, J.M. and Chang, P., "Optimizing the Performance of a Relational Algebra Data Base Interface", Comm. ACM, Vol. 18, No. 10, October 1975.

STEW74 Stewart, J. and Goldman, J., "The Relational Data Management System: A Perspective", Proc. ACM-SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May 1-3, 1974.

STON74 Stonebraker, M., "A Functional View of Data Independence", Proc. ACM-SIGMOD Workshop on Data

Description, Access, and Control, Ann Arbor, Michigan, May 1-3, 1974.

STON74a Stonebraker, M. and Wong, E., "Access Control in a Relational Data Base Management System by Query Modification", Univ. of California, Berkeley, ERL Memo No. ERL-M438, May 1974.

STON74b Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Management Systems", Univ. of California, Berkeley, ERL Memo No. ERL-M473, August 1974.

STON76 Stonebraker, M., Wong, E., Kreps, P. and Held, G., "The Design and Implementation of INGRES", ACM Trans. on Database Systems, Vol. 1, No. 3, pp. 189-222, September 1976.

TODD75 Todd, S.J.P., "Peterlee Relational Test Vehicle PRTV, A Technical Overview", IBM Scientific Centre Report UKSC 0075, Peterlee, England, July 1975.

TODD75a Todd, S.J.P., "PRTV, an Efficient Implementation for Large Relational Data Bases", Proc. International Conference on Very Large Data Bases, Framingham, Mass., September 22-24, 1975.

TODD76 Todd, S.J.P., "Integrated Architecture for Transaction Specification and Optimization in Relational Data Base Systems", IBM Scientific Centre Report UKSC 0085, Peterlee, England, November 1976.

TSIC77 Tsichritzis, D.C. and Lochovsky, F.H., DATA BASE

MANAGEMENT SYSTEMS, Academic Press, New York, 1977.

WONG76 Wong, E. and Youssefi, K., "Decomposition - A Strategy for Query Processing", ACM Trans. on Database Systems, Vol. 1, No. 3, pp. 223-241, September 1976.

RELATED BIBLIOGRAPHY

ALLE71	Allen, F.E., "A Basis for Program Optimization", Proc. IFIP Congress 1971, pp. 385-390.
ARMS74	Armstrong, W.W., "Dependency Structures of Data Base Relationships", Information Processing 74 (Proc. IFIP Congress, Stockholm, Sweden, August 5-10, 1974), North-Holland, Amsterdam.
ASH68	Ash, W. and Sibley, E.H., "TRAMP, An Interpretive Associative Processor with Deductive Capabilities", Proc. ACM 23rd Natl. Conf., Brandon Systems Press, Princeton, N.J., pp. 143-156, 1968.
BELL66	Bell, C.J., "A Relational Model for Information Retrieval and the Processing of Linguistic Data", IBM Research Report RC1705, Yorktown Heights, New York, November 1966.
BACH71	Bachman, P., "A Contribution to the Problem of the Optimization of Programs", Proc. IFIP Congress 1971, pp. 397-401.
BRAC72	Bracchi, G., Fedeli, A. and Paolini, P., "A Relational Data Base Management System", Proc. ACM National Conference, 1972, pp. 1080-1089.
COCK69	Cocke, J. and Schwartz, J.T., PROGRAMMING LANGUAGES AND THEIR COMPILERS, Preliminary Notes, Courant Inst. of Math. Science, New York, 1969.

CODD74 Codd, E.F., "Recent Investigations in Relational Data Base Systems", Information Processing 74, North Holland, Amsterdam.

COLL72 Collmeyer, A., "Implications of Data Independence on the Architecture of Data Base Management Systems", Proc. 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control, Denver, Colo., Nov. 1972.

DATE71 Date, C.J. and Hopewell, P., "File Definition and Logical Data Independence", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Cal., Nov. 1971.

DATE71a Date, C.J. and Hopewell, P., "Storage Structure and Physical Data Independence", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Cal., Nov. 1971.

DELO73 Delobel, C. and Casey, R.G., "Decomposition of a Data Base and the Theory of Boolean Switching Functions", IBM Journal of Research and Development, Vol. 17, No. 5, September 1973.

EPST77 Epstein, R., "A Tutorial on INGRES", Univ. of California, Berkeley, ERL Memo No. UCB/ERL M77/25, April 1977.

GRIE71 Gries, D., "Compiler Construction for Digital Computers", Wiley and Sons, New York 1971.

HEAT71 Heath, I.J., "Unacceptable File Operations in a

Relational Data Base", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Cal., Nov. 1971.

HAER76 Haerder, T., "An Implementation Technique for A Generalized Access Path Structure", IBM San Jose Research Report RJ1837, October 1976.

HOPK71 Hopkins, M., "An Optimizing Compiler Design", Proc. 1971 IFIP Congress.

JORD75 Jordan, D.E., "Implementing Production Systems with Relational Data Bases", Proc. ACM-PACIFIC 75 Conf., pp. 39-43, April 1975.

LEVI67 Levien, R.E. and Maron, M.E., "A Computer System for Inference Execution and Data Retrieval", CACM Vol. 10, No. 11, pp. 715-721, Nov. 1967.

LEVI69 Levien, R.E., "Relational Data File: Experience with a System for Propositional Data Storage and Inference Execution", Rand Corp. Mem. RM-5947-PR, Santa Monica, Cal., April 1969.

LUCK74 Lucking, J., "Data Base Languages, in particular DDL Development at CODASYL", Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

LUM73 Lum, V.Y., "General Performance Analysis of Key-to-Address Transformation Methods Using an Abstract File Concept", CACM Vol. 16, No. 10, pp. 603-612, Oct. 1973.

MANA75	Manacher, G.K., "On the Feasibility of Implementing a Large Relational Data Base with Optimal Performance on a Minicomputer", Proc. International Conference on Very Large Data Bases, Framingham, Mass., September 22-24, 1975.
MCDO75	McDonald, N., "CUPID: A Graphics Oriented Facility for Support of Non-Programmer Interactions with a Data Base", Ph.D. Dissertation, Univ. of California, Berkeley, ERL Memo No. ERL-M563, 1975.
MCLE75	McLeod, D.J. and Meldman, M.J., "RISS: A Generalized Minicomputer Relational Data Base Management System", Proc. National Computer Conference, Anaheim, Calif., May 19-22, 1975.
OLLE68	Olle, T.W., "A Non-Procedural Language for Retrieving Information from Data Bases", IFIP Congress, Edinburgh, North-Holland, Amsterdam, Aug. 1968.
PARK72	Parker, J.L. and Jervis, B., "An Approach for A Working Relational Data System", Proc. 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control, Denver, Colo., Nov. 1972.
RAMI71	"RAMIS - Users Manual", Mathematica, Inc., Princeton, New Jersey.
RISS73	Rissanen, J. and Delobel, C., "Decomposition of Files, A Basis for Data Storage and Retrieval", IBM San Jose Research Report RJ-1220, May 1973.
TSIC77	Tsichritzis, D. (ed.), "A Panache of DBMS Ideas",

Technical Report CSRG-78, Computer Systems Research Group, Univ. of Toronto, Toronto, Canada, February 1977.

WHIT72 Whitney, V.K.M., "RDMS: A Relational Data Management System", Proc. Fourth International Symposium on Computer and Information Sciences (COINS IV), Miami Beach, Florida, December 14-16, 1972, Plenum Press.

WHIT74 Whitney, V.K.M., "Relational Data Management Implementation Techniques", Proc. ACM-SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May 1-3, 1974.

YOUS77 Youssefi, K., et al, "INGRES Reference Manual - Version 6", Univ. of California, Berkeley, ERL Memo No. ERL-M579, April 1977.