QUERY PROCESSING IN A RELATIONAL DATABASE MANAGEMENT SYSTEM

by

K. Youssefi and E. Wong

Memorandum No. UCB/ERL M78/17

1 March 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Query Processing in a Relational Database Management System

Karel Youssefi and Eugene Wong
University of California, Berkeley

1. Introduction

In [WONG 76] we presented a general strategy for query processing known as decomposition , which had been designed for the nonprocedural query language QUEL [HELD 75] and was in the process of being implemented in the relational database management system INGRES [STONEBRAKER 76]. In this approach the problem of dealing with a multi-relational query is separated into two stages. First, a query which references several relations is decomposed into simpler components, the main objective at this point being to minimize the combinatorial growth that multi-relational queries entail. The information used to achieve this minimization consists mainly of the structure of the query and size statistics of the relations used. Once the query is decomposed into "sufficiently simple" components, the focus of the strategy shifts from one of structural simplification to that of minimizing data access, and in this "end game" phase of query processing the information associated with storage structure plays a dominant role.

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

One of the principal tactics that we proposed for breaking up a query into simple pieces was reduction , a process which might be described as separating a query at its natural joints. On both intuitive and theoretical grounds reduction appeared to be a highly advantageous tactic. However, the overall procedure of decomposition was too complex to permit a complete theoretical analysis, and the efficacy of reduction required confirmation by an empirical study. Such a study has now been undertaken [YOUSSEFI 78], and a summary of its major findings will be reported in this paper.

A second issue to be addressed in this paper is the strategy of the "end game". In contrast to decomposition, a considerable amount of work has already been published on strategies for processing two-variable queries. Our work differs from these both in the way the problem is abstracted and in the specific assumptions concerning the costs that the implementation environment imposes.

In section 2 we shall review the principal features of the decomposition procedures with an emphasis on the tactic of reduction. In section 3 the results of our experiments concerning reduction will be summarized. In section 4 details of the end game strategy together with a summary of some empirical studies will be presented.

2. Decomposition

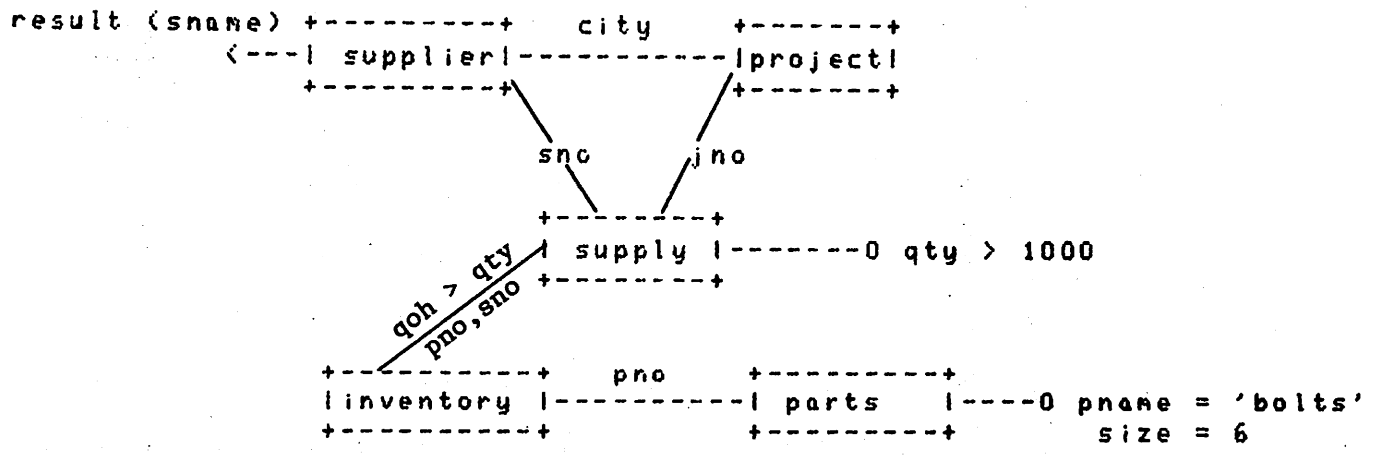
The principal features of decomposition are easily explained by an example. Consider a database which contains the following relations with the domain names appearing within the parenthesis:

```
supplier (sno, sname, city)
parts (pno, pname, size)
project (jno, jname, city)
inventory (sno, pno, qoh)
supply (sno, jno, pno, qty)
```

Suppose that we want to find "the names of all those suppliers each of whom supplies #6 bolts to some project located in the same city in quantity greater than 1000 but less than its quantity-on-hand". This query can be expressed in QUEL as follows:

```
range of s is supplier
range of p is parts
range of j is projects
range of v is inventory
range of y is supply
retrieve into result (s.sname)
where (s.sno = y.sno) and (s.city = j.city)
and (j.jno = y.jno) and (v.pno = y.pno)
and (j.qty < v.qoh) and (v.pno = p.pno)
and (p.pname = 'bolts') and (p.size = 6)
```

The structure of this query is revealed by the following self-explanatory graphical representation: [WONG 77]



The following tactics for decomposition have been proposed:

(a) Query Transformation - The query can be transformed into a number of semantically equivalent forms. Of these, the addition of one-variable conditions by invoking transitivity is probably the most useful. In our example, the conditions (v.qoh > y.qty) and (y.qty > 1000) imply that (v.qoh > 1000) can be added as a condition. Since every added restriction means smaller intermediate results, one-variable conditions due to transitivity should always be added.

(b) Reduction - A query is said to be reducible if it can be separated into two pieces which overlap on a single variable and each of which has at least two variables. An irreducible query is one that is not reducible. In [WONG 76] we showed that the irreducible components of a query are unique. For our example, they can be represented graphically as follows:

```
qoh>1000 +-----------+  pno   +-------+
  0-----| inventory |-------| parts |----0  pname = 'bolts'
         +-----------+        +-------+        size = 6
               |
               v
      newinv (sno,pno,qoh)

          +--------+ qoh > qty +--------+
          | newinv |-----------| supply |-------0 qty > 1000
          +--------+  pno,sno  +--------+
                                    |
                                    v
                          newsupply (sno,jno)

                          +-----------+
                         /| newsupply |
                        / +-----------+
                  sno  /        |
                      /         | jno
result (sname) +-/---------+    +---------+
       <----| supplier |-------| project |
            +----------+ city  +---------+
```

Thus, there are three irreducible components in our example, one with three variables and two with two variables.

(c) One-Variable Subquery Processing - Often, one or more relations referenced by a query can be reduced in size by both restriction (fewer rows) and projection (fewer columns). Such an operation corresponds to a one-variable subquery which can be detached and processed separately. In our example, there are three such one-variable subqueries, and they can be represented as follows:

```
pname = 'bolts'   +---------+
      0-----------| parts   |---------> newparts (pno)
 size = 6         +---------+


qty > 1000        +---------+
     0------------| supply  |---------> supply (pno,sno,jno,qty)
                  +---------+


qoh > 1000        +-----------+
     0------------| inventory |-------> inventory (sno,pno,qoh)
                  +-----------+
```

(d) Tuple-Substitution - Suppose that one of the variables in a query is replaced by the tuples in its range relation, tuple at a time. What results then is a collection of queries (one per substituted tuple) each with one less variable.

On intuitive grounds, the merits and demerits of each of these tactics can be evaluated. Tuple substitution is a tactic of last resort. Successive substitution for every variable but one in the query is tantamount to constructing a cartesian product of all the range relations. Such a procedure will always work, but its efficiency is deplorable. Reduction, on the other hand, prevents combinatorial growth at little processing cost and should probably be undertaken whenever possible. Whether or not a one-variable subquery should be detached depends on both the storage structure and the semantics of the query, and it is a decision that is best left to the end game.

The advantages of reduction were elaborated in [WONG 76] and can be briefly stated. It replaces a query by a sequence of queries with fewer variables and smaller (usually much smaller) relations. It allows the difficult decision of choosing a variable for tuple substitution to be deferred until a time when the choice is clearer and the potential for mistakes more limited.

One possible disadvantage of reduction is that it constrains the order in which the relations are accessed. The graphical representation of our example makes this clear. Once reduced, the query consists of three irreducible components which must be processed in the following sequence:

(parts,inventory)--->(newinv,supply)--->(newsupply,project,supplier)

It is conceivable that in some situations such a restriction on the order of processing would prevent an optimum strategy from being realized. Of course, such an optimum strategy may be difficult to discover and may not be realized anyway. Nonetheless, our hypothesis that the constraint on sequential order imposed by reduction rarely entails a major sacrifice in efficiency required empirical verification.

A second possible disadvantage of reduction is that if a joining variable is to be substituted it would appear to be better done prior to reduction, since then a single substitution simultaneously reduces the number of variables in two components. However, a separate substitution for the joining variable in the second component may be an advantage rather than a disadvantage since its range will have been reduced after the first component is processed.

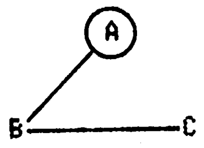
In sum, we found it impossible to completely evaluate the effects of reduction by a theoretical analysis. Thus, a series of experiments were constructed for an empirical assessment.

3. Experiment on Reduction

Our experiment consisted of running a series of multivariable queries, each of which was free of one-variable subqueries, using every possible strategy which combined reduction and tuple substitution. For each run the total number of data pages accessed was measured, and the strategies were compared solely on the number of pages accessed. Although there are other costs in processing a query, the most significant component for a complex query is the time for accessing the data. Furthermore, other costs such as overhead do not differ significantly from one strategy to another.

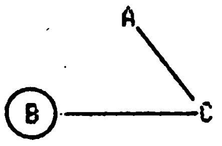
A collection of 11 queries were chosen. These range in the number of variables from 3 to 4, and in the sizes of the relations from 15 tuples (2 pages) to 138 tuples (70 pages). The structure of these queries together with the sizes of the relations involved can be represented graphically. Let each relation be represented by a node in a graph, and let an arc between two nodes signify the existence of a conjunction clause in which both relations are referenced. The circled nodes are those which appear in the target list (i.e., in the relations referenced by the result). The queries for which measurements were made can be represented as follows:

Query 1



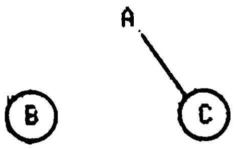
```
tA = 66   PA = 15
tB = 56   PB = 13
tC = 51   PC = 14
```

Query 2



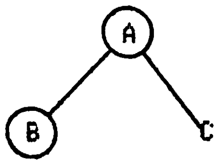
```
tA = 66   PA = 15
tB = 56   PB = 13
tC = 150  PC = 39
```

Query 3



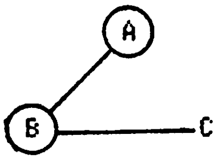
```
tA = 66   PA = 15
tB = 56   PB = 13
tC = 150  PC = 39
```

Query 4



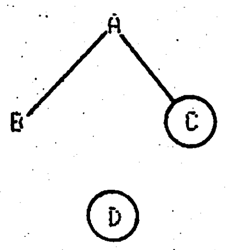
```
tA = 62   PA = 32
tB = 62   PB = 17
tC = 56   PC = 29
```

Query 5



```
tA = 66   PA = 15
tB = 51   PB = 14
tC = 56   PC = 13
```

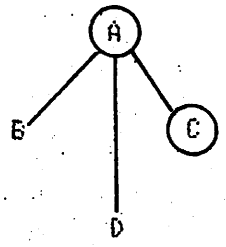
Query 6



tA = 98   PA = 50
tB = 79   PB = 17

tC = 62   PC = 17
tD = 40   PD = 3

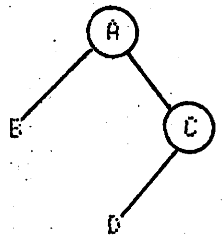
Query 7



tA = 56   PA = 29
tB = 62   PB = 17

tC = 112 PC = 57
tD = 97   PD = 26

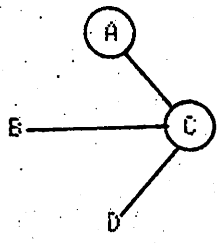
Query 8



tA = 62   PA = 32
tB = 51   PB = 14

tC = 56   PC = 29
tD = 62   PD = 17

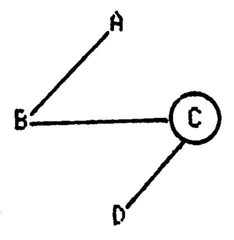
Query 9



tA = 62   PA = 32
tB = 62   PB = 17

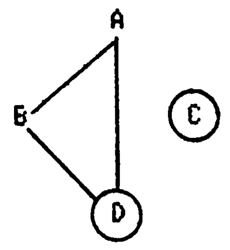
tC = 138 PC = 70
tD = 51   PD = 14

Query 10



tA = 62	PA = 17
tB = 51	PB = 14
tC = 62	PC = 32
tD = 56	PD = 29

Query 11



tA = 51	PA = 14
tB = 56	PB = 29
tC = 62	PC = 17
tD = 62	PD = 32

The queries used in our experiment were free of one-variable subqueries. Hence, the first move in any strategy must be either a tuple-substitution or a reduction. We wanted answers to the following questions from our series of experiments:

(1) Since reduction always puts a component containing the target list near the end in the sequence of processing, is it often the case that a target list variable should be substituted for before reduction?

(2) Is it often the case that tuple-substitution for a joining variable should precede reduction?

(3) Is reduction as the first move a good general strategy?

In Table 3.1 we have tabulated the page counts for the best strategy of each of the following four categories:

(a) The first move is substitution for a target list variable.

(b) The first move is substitution for a joining variable.

(c) The first move is tuple substitution without restriction on the variable.

(d) The first move is reduction.

For each case page counts for both keyed and non-keyed storage structures are presented.

	(a) target-list		(b) joining		(c) best-tuple-subs		(d) reduction	
query	keyed	non-keyed	keyed	non-keyed	keyed	non-keyed	keyed	non-keyed
1	892	980	255	793	255	793	208	744
2	1450	2265	479	2243	479	2243	263	1888
3	12713	144760	636	2261	636	2261	319	1944
4	540	1986	157	1083	157	1083	354	354
5	2054	2458	469	1015	469	1015	323	578
6	51420	128083	1637	3015	1637	3015	519	519
7	3357	3670	3357	3670	3357	3670	643	957
8	2230	4257	2230	4257	2230	4257	471	1302
9	88470	113296	3145	4329	3145	4329	3042	4458
10	357	927	357	927	357	927	419	655
11	34038	112716	14751	14751	14751	14751	645	1914

Table 3.1 Reduction vs. Tuple Substitution

The implications of our experiments seem rather clear. First, reduction as the first move is the best in every single case. The difference in some cases (e.g., 11) is rather dramatic. Second, if tuple substitution is to be selected as the first move, then the best choice in every case is a joining variable. Finally, the fact that reduction defers the access for the target list variables till the end had no unfavorable effect in any of the queries of our experiment. Indeed, tuple substitution for a target list variable as a first move often had disastrous consequences. (See e.g., queries 3,6,9,11.)

We believe that the efficacy of reduction as a decomposition tactic has been well demonstrated by evidence. It is the one tactic which is most responsible for the prevention of combinatorial growth.

4. The End Game

Decomposition in general and reduction in particular provide a means for reducing the complexity of a query to a point when the effects of storage structure on access efficiency become clearer. An appropriate point at which to stop and focus our attention on the storage structure appears to be when the number of variables is two or less. Indeed, two variable queries have received a great deal of attention in the literature [ROTHNIE 75, BLASGEN 75]. An effort has been made in [YAO 78] to organize the many techniques which have been suggested into a coherent theory. The focus of our attention is somewhat different, our objectives being:

(a) to provide a detailed evaluation of two-variable strategies in the specific context of the INGRES system, and

(b) to present some experimental results on these strategies.

In INGRES access to stored data is deferred until a one-variable query is encountered. Thus, faced with a two-variable query, we have to make the following decisions.

(a) If a one-variable subquery involving restriction and/or projection on one of the relations can be detached, should it be detached and processed before the rest of the query?

(b) Since tuple substitution must be made for one of the two variables, how should the selection be made?

(c) Should the storage structure for one or both of the relations be modified?

An example will make these issues clear.

Consider the following two-variable query which is the first irreducible component of the example in section 2.

```
qoh > 1000  +-----------+     pno     +--------+
   0-------| inventory |-----------| parts  |------0
            +-----------+             +--------+    pname = 'bolts'
                  ↓                                  size = 6

           newinv (sno,pno,qoh)
```

Faced with such a query, we have to decide whether to take each of the following possible actions:

(a) We can restrict parts according to the condition (pname = 'bolts') and (size = 6) and project the result on pno before further processing.

(b) We can restrict inventory on (qoh > 100) before further processing.

(c) We can tuple substitute for parts or for inventory.

(d) We can modify the storage structure of parts or inventory.

All these decisions are affected by the existing storage structures, and to determine how they are affected, we need to abstract the relevant factors of the storage structures.

Let R be a relation and let B(r), r R, be a truth function on R. Let q(R,B) be the number of tuples in R which satisfy B. Let t(R,B) be the number of tuples which have to be examined to determine the qualifying ones, and P(R,B) be the number of pages which have to be accessed to determine the qualifying ones. Storage structures potentially achieve one or both of two things: (a) Keep t(R,B) as close to q(R,B) as possible. (b) Minimize P(R,B). If $t(R,B) \cong q(R,B)$, we shall say that R is indexed on B, and if $F(R,B) \cong q(R,B)\cdot$(width of R/page size), we shall say R is clustered on B.

We observe that for R to be clustered on B it is necessarily indexed on B, but the converse is not true. Typically, a secondary index on a domain "indexes" a relation on any condition referencing that domain, but does not cluster it unless the condition is satisfied by at most one tuple.

As an example consider the relation parts (pno,pname,size) "hashed on pno" with a secondary index on pname. We shall assume that to a given pno corresponds at most a single tuple, but that there may be several tuples with the same pname. Consider the conditions

B1 : pno = 12345

B2 : pname = 'bolts'

The relation parts is indexed on both B1 and B2, but clustered only on B1 and not B2. However, suppose that the roles of pno and pname are reversed in the storage structure, i.e., parts is hashed on pname with a secondary index on pno. Then parts would be clustered on B2 because the primary structure is keyed on pname, and also clustered on B1 because there is a secondary index on pno and B1 is satisfied by at most one tuple.

The first issue in the end game that we shall try to resolve is the detachment of one-variable subqueries. A general two-variable query in QUEL has the form:

```
range of r is R
range of s is S
retrieve into result name (TL(r,s)) where qualification
```

TL denotes the target list (i.e., the domains of the result relation), and the qualification has the general form:

```
C1 (r) and C2 (s) and C(r,s)
```

where C1 and C2 are one-variable clauses which can be detached.

Intuitively, the cost of processing a one-variable subquery is linear in the cardinality of the relation involved, while the cost of processing a two-variable query is super-linear in the sum of the cardinalities. This line of reasoning suggests that it is nearly always advantageous to restrict R on C1 and S on C2. There is one important exception to this rule. Suppose that R is not clustered on C1 (r) but R is clustered on C(r,s) for every s.

Then access to R is best deferred until a variable is chosen for tuple-substitution because should s be substituted, then access to R using C(r,s) and C1 (r) together is far more efficient than access using C1 (r) alone.

Consider our example:

```
qoh > 1000      +-------------+  pno  +---------+        pname = 'bolts'
      0---------| inventory   |-------| parts   |-------0
                +-------------+       +---------+          size = 6
                      ↓

                newinv (sno,pno,qoh)
```

Assume that inventory is stored in a keyed-structure with (pno,sno) as key and parts is hashed on pname. Then, the one-variable subquery

```
               pname = 'bolts'    +--------+
                  0---------------| parts  |-----> newparts (pno)
                size = 6          +--------+
```

clearly should be detached, but the one- variable subquery

```
     qoh > 1000   +-----------+
      0-----------| inventory |-------> inventory' (sno,pno,qoh)
                  +-----------+
```

should not be detached. Access to inventory is best made using the condition on pno rather than qoh.

The second decision that we have to face is the selection of a variable for substitution. Let Q(r,s) denote a two-variable query, TL(r,s) its target list and C(r,s) its qualification. Let |R| and |S| denote the cardinality of R and S respectively. Suppose that we substitute for r, then we get |R| one-variable queries on S. The total cost measured in pages accessed is

$$\text{cost (substitute for r)} = |R| + |R|\,P(S,Q)$$

where P(S,Q) is the average number of pages accessed to process Q for each substituted r. It follows that

$$\text{cost}_{\min}(Q) = \min(\,|R| + |R|\,P(S,Q),\ |S| + |S|\,P(R,Q))$$

and the variable chosen for substitution should correspond to the smaller of the pair

$$\frac{|R|}{P(R,Q) + 1}, \qquad \frac{|S|}{P(S,Q) + 1}$$

The difficulty in using this decision rule is that while |R| and |S| can be assumed known, P(R,Q) and P(S,Q) must be estimated. Of the factors which affect P(R,Q), the following are probably the most important:

(1) P(R), the number of pages occupied by R.

(2) Whether or not R is referenced by the target list.

(3) Whether or not R is clustered relative to the qualification.

The reason that (2) is important is because in QUEL there is an implied existential quantifier on any variable not referenced in the target list. Hence, in a query (TL(s), C(r,s)) only a single r tuple satisfying C needs to be found for each substituted s.

The simplest situation arises when neither R nor S has a useful storage structure and both appear in the target list. Then, P(R,Q) = P(R), P(S,Q) = P(S) and both are known quantities. In Table 4.1 we have summarized the experimental data on eight queries of this kind.

query	|R|/P(R)+1	|S|/P(S)+1	pages (R substituted)	pages (S substituted)
1	1.944	4.125	2104	4633
2	3.444	1.944	4307	2355
3	4.125	5.789	3709	4918
4	3.444	1.931	3538	1950
5	4.125	4.389	1135	1194
6	4.556	4.647	1329	1358
7	3.565	10.125	595	1788
8	1.9375	11.515	2014	11811

Table 4.1

We see that not only is our criterion verified in every case, but the cost is very nearly proportional to the ratio #tuples/(#pages + 1).

The situation becomes more difficult to analyze if only one of the two relations (say R) is referenced in the target list. In that case P(S,Q) is very likely to be less than P(S) since we only need to find one tuple in S which satisfies Q. In Table 4.2 we compare the measured values for P(S,Q) with P(S) for 8 queries in which S is not referenced in the target list

Q	P(S)	P(S,Q)	ratio
1	70	2.2	.03
2	70	63.25	.90
3	15	5.33	.35
4	57	11.82	.21
5	15	5.58	.37
6	29	2.48	.08
7	14	10.75	.77
8	29	26.78	.92

Table 4.2 Relation Not Referenced in Target List

Unfortunately, the results afford us little predictive value for P(S,Q). It is always less than P(S), but beyond that we need additional information about the data to make a useful prediction for P(S,Q).

The situation is equally murky if one or both of the relations has a useful storage structure. Suppose that S is clustered on C(r,s) for each fixed r. This means that for a fixed r the s-tuples which satisfy C(r,s) are on nearly the minimum number of pages. However, if the number of such s-tuples is not known, estimating P(S,Q) is still difficult. In Table 4.3 we again compare the measured P(S,Q) with P(S), but this time S is always referenced in the target list and is clustered with respect to Q.

Q	P(S)	P(S,Q)	ratio
1	70	53.2	.76
2	70	2.3	.03
3	15	11.4	.76
4	57	44	.77
5	15	9.8	.65
6	17	14	.82
7	22	1.1	.05
8	31	1.4	.05

Table 4.3 Effect of Keyed Structure

As in the case of a relation not referenced in the target list, our experimental results afford us little predictive value.

Suppose that the condtion C(r,s) is of the simple form

$$r.A = s.B$$

where A is a domain in R and B a domain in S, and suppose that we know the quantity

k = number of different A values in R

Then, we can make a reasonable estimate of P(S,Q) when S is not referenced in the target list or when S has a keyed storage structure with B as its key.

For each r, there are on the average |S|/k tuples in S which satisfy C(r,s). If S is not referenced in the target list, only one of these tuples need be found and the average number of pages that would have to be accessed is

(a) $P(S,Q) = P(S)/(k|S|)$

On the other hand if S is keyed then the number of pages that would have to be accessed is

(b) $P(S,Q) = (|S|/k)$ / # tuples per page
$= P(S)/k$

Our experiments provided a limited amount of data for verifying these formulas. In Table 4.4 we have summarized the results on five queries, each of which involving two relations R and S such that S is not in the target list while R is keyed. Thus, the same set of queries can be used to check both of our estimation formulas.

	P(R,Q)		P(S,Q)	
query	estimated	measured	estimated	measured
1	8	12	1	1
2	1	1	30	62
3	56	42	2	5
4	8	13	4	11
5	3	12	2	4

Table 4.4 Verification of Estimates

The estimates erred consistently on the low side, but on the whole the agreement was not bad for such simple formulas. What is more significant is that if these estimates had been used in comparing |R|/(P(R,Q) + 1) and |S|/(P(S,Q) + 1) to select a variable for substitution, the correct decision would have been made in every case, and by a comfortable margin.

Tuple substitution for one variable in a two variable query results in a collection of similar queries on the remaining variable. Since the collection is homogeneous in the form of the qualifying condition, the same storage structure would allow the relation to be clustered for every query in the collection. Specifically, suppose that we substitute for r in R. Then, if S is clustered with respect to C(r,s) for one r it is clustered for every r. Thus, if S is not already clustered with respect to C(R,s) it would probably pay to modify its storage structure. There are three choices in INGRES: modify to ISAM, modify to HASH, create secondary index. The cost of creating a secondary index is always less than modifying the primary structure, but the benefit is also less and sometimes down right useless. The relative cost of modification between ISAM and HASH is very much a function of the particular implementation environment. Since ISAM involves sorting, its cost must be super linear in terms of the relation size, but surprisingly it is often less than the cost of hashing. As implemented, the costs in pages accessed for modifying to HASH and ISAM are:

$$\text{cost (HASH)} \cong P(S) + 2.888\ |S|$$

$$\text{cost (ISAM)} \cong 7.5\ P(S) + 2P(S)\ \log 7\ (P(S)/64)$$

In Table 4.5 we have summarized some empirical results on storage structure modification. For each of 11 two-variable queries, we compare the total cost (in pages accessed) of processing using each of four strategies:

(a) no storage modification
(b) modify S to HASH after substitution for r
(c) modify S to ISAM after substitution for r
(d) create secondary index after substitution for r

The cost includes that of any storage modification which was undertaken.

query	(a) no modification	(b) HASH	(c) ISAM	(d) index
1	8748	2101	1356	1026
2	1196	622	972	10478
3	1710	806	937	7654
4	1191	299	500	1204
5	5400	703	536	2181
6	1196	451	404	10478
7	3960	1666	1504	2184
8	17820	1999	1955	2789
9	16170	1976	1486	2255
10	17820	---	916	22595
11	16170	---	504	10344

Table 4.5 Comparison of Storage Structure Modification

We note that some general conclusions can be drawn. First, in every case storage modification was useful and in several instances critical. Second, ISAM was the best structure overall, and for queries which involve inequalities in their qualification (e.g., queries 10 and 11) choosing ISAM is of vital importance. Finally, secondary index is rarely the best strategy (only query 1) and often it is down right dangerous (e.g., 2 and 6). All in all, once tuple substitution is undertaken modifying the remaining relation to ISAM appears to be the clear-cut winning strategy.

5. Conclusion

This is a follow up to our earlier paper [WONG 76] on query processing in the INGRES system. Our two principal goals have been: (a) to verify through an empirical study that "reduction" as a tactic in decomposition is of near panacea quality, and (b) to provide a detailed treatment, by both analytical and empirical means, of the "end game" phase of query processing. Although our experiments have been far from exhaustive, the consistency of the results leads us to belive that the major issues which we have posed are now resolved.

REFERENCES

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.

HELD75 Held, G., Stonebraker, M., and Wong, E., "INGRES - A relational data base system", Proc. 1975 National Computer Conference, pp.409-416.

ROTH 75 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.

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.198-222, September 1976.

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.

WONG77 Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, pp.217-235, May 1977.

YAO 78 Yao, S.B., "Optimization of Query Evaluation Algorithms", in press, 1978.

YOUS78 Youssefi, K.A., "Query Processing for a Relational Database System", Univ. of California, Berkeley, ERL Memo No.

UCB/ERL M78/3, January 1978.