LBL-11298 c. 2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

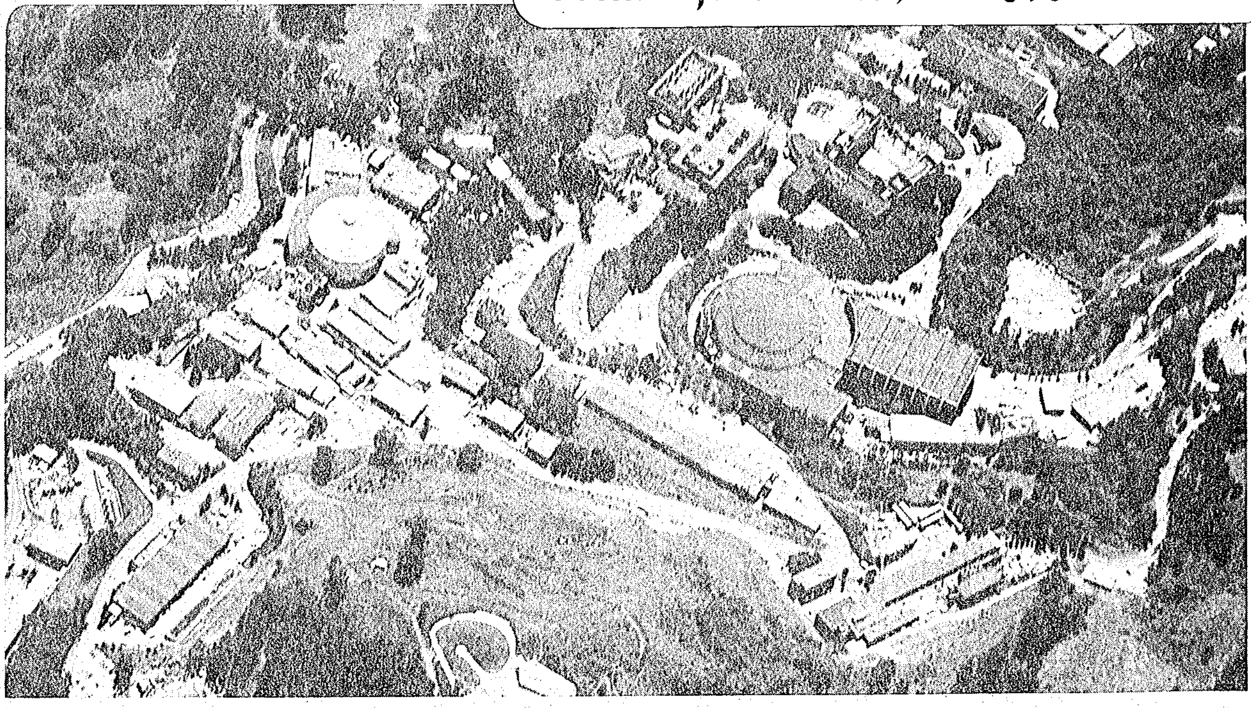
Physics, Computer Science & Mathematics Division

To be presented at the International Computer Symposium on Software Development: Methodology and Management, National Chiao Tung University, in cooperation with Academia Sinica, Taipei, Taiwan, Rep. of China, December 16-18, 1980

HIGH LEVEL PERFORMANCE ESTIMATE OF RELATIONAL QUERIES

Harry K.T. Wong

August 1980



LBL-11298 c. 2

Prepared for the U.S. Department of Energy under Contract W-7405-ENG-48

High Level Performance Estimate of Relational Queries

by

Harry K. T. Wong

Department of Computer Science & Applied Mathematics
Lawrence Berkeley Laboratory
University of California
Berkeley, California

Abstract

Performance estimate of queries is a necessary part of any efficient database design methodology. In this paper, a high level performance estimator for relational queries is presented which is different from conventional evaluators such as SYSTEM R [Selinger 1979] in that performance is predicted without the details of the low level constructs such as links and indices. Rather, abstractions and reasonable assumptions of these low level constructs are used in a set of formulas to estimate the performance of a set of queries against a schema.

The major results of the paper are (1) The realization and motivation of the need of high level performance estimators of this kind, and (2) a fast way to estimate the cost of N-way joins. The second result is interesting in that the algorithm for N-way joins is found to be similar to matrix multiplication optimization, with straight forward

This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research U.S Department of Energy under contract W-7405-ENG-48.

extensions.

1. Introduction and Background

Performance estimate of queries is a necessary part of any efficient database design methodology. In this paper, a high level performance estimator for relational queries is described which is different from conventional evaluators such as SYSTEM R [Selinger 1979] in that it attempts to perform cost estimates without the details of the low level constructs such as links and indices. Rather, abstractions and reasonable assumptions of these low level constructs are used to estimate performance of a set of queries against a schema.

The motivation of this research comes from a project on relational database design [Wong and Shu 1980]. Some understanding of our database design approach will help to motivate the need of high level evaluators.

Our database design approach is based on the conviction that semantics and efficiency are two inseparable, crucial database design issues. The semantics of a database is usually expressed via a set of legality conditions, called semantic integrity constraints. Examples are functional dependencies, subset-superset relationships, cardinality of relationships, etc. Our database design methodology hinges on a technique that can 'trade' semantics with performance. More specifically, a method is developed so that given a set of processes (representing the application dynamics) and a set of integrity constraints (representing the semantics of the application), an additional set of processes will be generated which are required to enforce the integrity constraints when the original set of processes is active in the application. This extra set of processes will represent the effort needed to enforce the integrity constraints. A simple example here will clarify the approach:

Given a relation

PRODUCT (PNO, VENDOR, PRICE, PNAME),

assume that we have an integrity constraint on PRODUCT

PNO -> PNAME

which asserts that a given value from PNO uniquely determines a value from PNAME. The following process

```
insert (p,s,50,'xyz') into PRODUCT
```

requires the enforcement of the functionality PNO -> PNAME. The following enforcement process is generated along with several alternatives as an extra process to the original insert

```
update PRODUCT
set PNAME = 'xyz'
where PNO = p
```

The meaning of this statement is that the function PNO -> PNAME is maintained by updating all the ranges of p to the value 'xyz'.

Now, the 'goodness' of a schema candidate can be measured by the efficiency of running the original set of processes and the set of the generated processess. That is, each time a schema is 'tuned' , in addition to the efficiency of the original processes, the efficiency of the generated processes is also subject to optimization.

With this technique, database design is transformed into a more technically tractable problem: the optimization of the performance of two sets of processes. The second set of processes (corresponding to the enforcement of constraints) is dependent on how the schema is changed, so this optimization of performance is <u>not</u> the same optimization problem in traditional physical database design, which is concerned with a <u>single</u> set of processes throughout the design procedure. Our objective is to derive a schema so that the cost of running the original processes and the extra processes is smallest.

Coming back to our example above, if insertion activities on relation PRODUCT are done often, the frequency of running the extra process is large and the cost of running it is large. An alternative in our methodology will consider is to break PRODUCT into two relations:

PRODUCT1 (<u>PNO</u>, <u>VENDOR</u>, PRICE)
PRODUCT2 (<u>PNO</u>, PNAME)

And now the extra process is no longer necessary since the functionality PNO -> PNAME is now expressed directly in PRODUCT2 and its enforcement is part of the underlying DBMS. However, this breaking up of PRODUCT may increase the frequency of an expensive activity: join of PRODUCT1 and PRODUCT2. If this need of join is in frequent demand, the decision of breaking up PRODUCT may be negated depending on the estimated savings.

Given a relation schema R, a set of processes P and a set of integrity constraints I, a methodology (which is reported in [Wong80]) is used to generate a set of processes P' which is required to enforce I when P is active. If cost is a function of efficiency estimate of processes, the problem of database design can now be stated as:

Derive a schema R so that cost (P) + cost (P') is minimal.

The meaning of this statement is that database design is a tradeoff between performance and integrity. If we modify schema R to reduce the cost of P, for example, we may (and usually will) introduce more integrity problems in R (such as unnormalized relations, update problems), but these integrity problems will be translated into extra processes in P', and the overall cost of P and P' may stop us from modifying the schema. Using the above statement as our objective, therefore, will guide us to derive a schema which is reasonable in performance and yet without intolerable (i.e., expensive to enforce) integrity problems. Contrary to traditional database design methods, our approach will allow a schema with integrity problems (such as having relations not in second normal form) as a candidate design as long as it is comparatively inexpensive to enforce or repair the integrity.

The methodology of generating processes from a set of processes to enforce integrity is described in [Wong 80].

Because of our approach, a tremendously large solution space of schemata is being explored for an efficient design. The cost of examining all possible physical designs for each schema candidate in order to evaluate the cost of a schema is prohibitively expensive. In the second half of this paper, we will describe a fast performance estimator used to implement the cost function mentioned above. In our evaluator approach, a high level model is designed so that only abstractions and reasonable assumptions of low level constructs such as links and indices are used to calculate an estimate of a schema. Section two reports an attempt to design such a performance model that can provide fast yet reasonable estimates of how efficient a schema is with respect to a set of queries.

A high level relational query language called SQL [Chamberlain 1976] will be used to express the set of processes that represent the operation of the application. It will be assumed that the reader is familiar with relational model concepts [Codd 1970].

2. Performance Estimate

In our model, we assume a simple storage structure of relations. Relations are stored consecutively in pages and they can be sorted in the order of any one of the columns. The unit of cost measure is the number of pages fetched from the secondary storage.

The following information on the schema is assumed to be available:

- The size of each relation in the schema (the estimated number of pages and tuples).
- The cardinality of all columns of each relation. The cardinality of a column A of relation R is the number of distinct values that can occur in column A in all tuples in R. For example, the age attribute of the PEOPLE relation may have a cardinality of 100. One further assumption is that the values of any column of any relation are evenly distributed. This means, for example, that one percent of the tuples in PEOPLE have the age value of 60. Rough as it is, this assumption provides a reasonable solution in simplifying the formulas in the model.
- The high and low values of every column in every relation. For example, 1 and 100 are respectively the low and high values of column age of relation PEOPLE.
- The maximum size (# of bytes) of every column of every relation.

The above information is actually derived from a requirement specification by the users of the applicaton. The requirement specification language is described in [Shu, Wong & Lum 1980] and the derivation of information is described in [Wong 1980 c].

A SQL query has the form

```
SELECT ...
FROM ...
WHERE Q
```

Q is a predicate expressing a condition which the resulting tuples from the query have to satisfy. To estimate the cost of a query, the 'selectivity factor' of Q will have to be estimated. The <u>selectivity</u> <u>factor</u> of Q is a number representing a fraction of the tuples predicted to satisfy Q. The selectivity factor of any predicate P can be computed from the components of P [Selinger 1979]. Below are some examples of selectivity factors of our model which are essentially that of [Selinger 1979].

Let SF stand for Selectivity Factor.

If Q has the form "column = Value", then SF = 1/(cardinality of column). This is the result of the even distribution assumption mentioned above.

If Q has the form "column BETWEEN value1, value2", then SF = (value2-value1)/(high value - low value). This is obtained through linear interpolation on values of a column.

If Q has the form "Q1 AND Q2", then SF = (SF of Q1) * (SF of Q2). Other example are covered in [Selinger79].

Given a query of the form

```
SELECT ...
FROM R1, R2, ... Rn
WHERE Q
```

Q is a predicate expressed in conjunctive normal form. We will use the notation SF(Rj) to stand for the selectivity factor of the conjuncts of Q which are "local" to Rj. A predicate is local to Rj if it involves only columns from Rj.

The following notation is used throughout.

P(R)	:	number of pages occupied by relation R
N(R)	:	number of tuples of relation R
L(R) or L(R,A)	:	length of tuples (# of bytes) of relation R or length of column A of R.
C(R,A)	:	Cardinality of column A of relation R

There are three main formulas in the model: single relation query, simple join and N-way joins.

<u>Simple relation query</u>: Consider a query of the following form

```
SELECT A, B, ...
FROM R
WHERE Q
ORDER BY J
```

The meaning of this query is that the tuples of relation R are scanned and those that satisfy the predicate Q will have their columns A,B, ... projected and presented in the order of column J.

If the relation R is sorted on column J, the cost would simply be the estimated number of pages fetched, i.e.

SF(R) * P(R)

If, however, sorting is needed, the cost SORTING(R,J)*SF(R) is added to the formula above. SORTING is a function to estimate the cost of sorting a relation on a column. It is similar to that of SYSTEM R and will not be presented here.

Simple joins : Given a join query of the form

```
SELECT A,B, ...
FROM R1, R2
WHERE Q1 AND R1.A = R2.A
```

First, if both R1 and R2 are sorted on column A, there are SF(R1) * P(R1) pages to be fetched from R1 for tuples estimated to satisfy the local predicate of R1 in Q1. It takes SF(R2) * P(R2) pages to find the matching tuples for the join from R2 that satisfy the local predicates of R2 in Q1, hence the cost is

SF(R1) * P(R1) + SF(R2) * P(R2)

If R1 or R2 is not sorted on A, the cost of SORTING(R1, A) or SORTING(R2, A) or both is added to the above formula.

N-way joins :

Given a join expression of the following form:

R1[A]R2[B]R3[C]R4 ...

where A, B, C, ... are join columns between relations R1, R2, R3, ... We need to decide the order of doing the joins so that the cost (# of pages fetched) is minimal. For example, the ordering

((R1[A]R2) [B] (R3[C]R4))

may be less expensive than

(((R1[A]R2) [B] R3) [C] R4)

even though the result is the same.

SYSTEM R considers the join order determination of N-way join. Starting with access paths to single relations, the method is to consider all combinations of doing 2-way, 3-way ... and N-way joins among the relations with pruning heuristics to cut down the search space. In this paper, a different method is used. First, because of the lack of access paths, the search space is smaller. Second, a more efficient way of organizing the search is found. The idea is that N-way join order determination is found to be similar to matrix multiplication where a sequence of submatrix multiplications is found so that the number of arithmetic operations is the smallest. The relations R1, R2, R3, R4 above can be thought of as the matrices involved and join columns A, B, C, D can be thought of as the dimensions of the matrices. A similar solution to the matrix multiplication optimization can be used to find the optimal ordering of join expressions. The technique used in the matrix multiplication is a common technique called Dynamic Programming [Aho et. al74] and can be used to compute the N-way join cost. Starting with the 2-way joins, we successively compute the cost of larger and larger segments using the intermediate results of the smaller segments. The larger segments themselves in turn are saved for yet larger segments in the join sequence. Let c be the cost function and c(i, j) stands for the cost of doing the join sequence Ri[]...[]Rj.

Computing c(i,j) for the segment

Ri[ ]...[ ]Rj

involves locating a k between i and j so that the expression:

c(i,k) + c(k+1,j) + cost of joining the two composite relations

is smallest. Since the results of the computations c(i,k) and c(k+1,j) have been saved, the best cutting point k can be efficiently found. In order to compute the cost of joining composite relations, we need to find the parameters of the joined relations (such as N(R) and P(R)). This is illustrated next.

We assume that the cardinalities of column A in the join expression R1[A]R2 are the same, denoted by C(A). First, the number of tuples in the join expression R1[A]R2 is the the product of the cardinalities of R1 and R2 times the product of their selectivity factors. To eliminate the duplicated tuples due to the repeated value of column A, we have to divide the result by the cardinality of column A. This gives us the formula:

$$N = (N(R1) * N(R2) * SF(R1) * SF(R2))/C(A)$$

The number of pages occupied by these N tuples is bounded by this number

$$P = N*(L(Rk) + L(R(k+1)) - L(Rk,A))/S$$

where S is the page size (number of bytes).

Given N and P, the cost of joining two composite relations c(i,k) and c(k+1, j) can be computed using the formulas for simple joins.

The following program describes the N-way join in a more precise manner. Note that the cost of joining relations Ri and R(i+1) is obtained from formulas listed above.

```
     do i = 1 until n
           c(i,i) = 0
     end
/* d is a variable for the size of the join segment*/
/* i points to the beginning of the join segment   */
/* j points to the end of the join segment and it  */
/* is always equal to i+d                          */
do d = 1 until n-1
          do i = 1 to n-d
               j = d + i
               c(i,j) = min(c(i,k) + c(k+1,j) + cost of
                         joining the two composite
                         relations) where i <= k < j
          end
end
```

N-way join program

This algorithm has a computational complexity of $O(n^3)$ where n is the number of relations in the join sequence.

3. Conclusion

In this paper, we have presented a high level performance estimator that provides fast estimates of the cost of a set of queries against a schema. The main goal of the paper is to point out the need of this kind of estimator where only the abstractions of low level constructs are used to predict the efficiency of queries at the schema level, not at the detailed physical structure level. Because the latter level involves a large number of possible physical structures for each schema, it is very expensive to evaluate the efficiency of a large number of schemata since a very large number of possible physical structures will have to be looked at in order to decide whether a given schema can (or cannot) have efficient physical design. This paper describes a high level evaluator for relational queries. This evaluator is used to give rough estimates for our database design methodology. Some interesting problems related to this research are listed below.

- Incorporating the abstract forms of lower level physical structures such as links (indexes) into the high level evaluator so that the evaluation can be done more accurately,
- validation of the evaluator so that each underlying assumption can be judged on its effect on cost estimates,
- examination of a similar evaluator for different data

models such as DBTG or IMS to see if it is possible to come up with a set of reasonable assumptions and fast formulas to predict efficiency, and

- consideration of more complicated query types such as nested queries, and update commands such as insert and delete.

Acknowledgement The author would like to thank Dr. Don Austin for his helpful comments. The careful and critical review of Dr. Arie Shoshani is very much appreciated.

Reference

Aho, A.V., Hopcroft, J.E., Ullman, J. D., The Design and Analysis of Computer Algorithms Addison-wesley, 1974.

Astraban, M.M. et al, "SYSTEM R: Relational Approach to Database Management", TODS Vol. 12, 1976.

Chamberlain, D.D. et al., "SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control," IBM J. of Research and Development, Vol. 20, #6, 1976.

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, #6, 1970.

Hammer, M., Niamir, B., "A Heuristic Approach to Attribute Partitioning," PROC. SIGMOD 79, Cambridge, MA.

Selinger, P., G. et al. "Access Path Selection in a Relational Database Management System," PROC. SIGMOD 79, Cambridge, MA.

Shu, N., Wong, H.K.T., LUM, V., "Forms Approach to Application Specification for Database Design," IBM Research Lab., RJ2687, 1980.

Wong, H.K.T., Shu, N., "An Approach to Relational Database Schema Design," IBM Research Lab., RJ 2688, 1980.

Wong, H.K.T.,"Semantics, Performance Tradeoff in Relational Database Design," to appear.

Wong, H.K.T., "Database Application Development Through Requirement Specification," to appear.