THE EFFECT OF JOIN SELECTIVITIES ON OPTIMAL NESTING ORDER

Akhil Kumar and Michael Stonebraker

EECS Department
University of California
Berkeley, Ca., 94720

Abstract

A heuristic query optimizer must choose the best way to process an incoming query. This choice is based on comparing the expected cost of many (or all) of the ways that a command might be processed. This expected cost calculation is determined by statistics on the sizes of the relations involved and the selectivities of the operations being performed. Of course, such estimates are subject to error, and in this paper we investigate the sensitivity of the best query plan to errors in the selectivity estimates. We treat the common case of join queries and show that the optimal plan for most queries is very insensitive to selectivity inaccuracies. Hence, there is little reason for a data manager to spend a lot of effort making accurate estimates of join selectivities.

1. INTRODUCTION

In a relational database system, relations are accessed by specifying queries in a query language such as SQL [IBM84] or QUEL [HELD75]. Each user command is processed by a heuristic query optimizer at compile time or at run time to select the minimum expected cost query execution plan. This plan is then executed to deliver the result to the user who submitted the command.

Consider, for example, the following selection query on the EMP relation:

```
retrieve (EMP.name)
where EMP.age > 50  and EMP.salary < 1000
```

If there are secondary indexes on both age and salary, then the optimizer must choose between 3 processing plans, namely scan the whole EMP relation, scan a portion of the age index and then access qualifying EMP records, or scan a portion of the salary index. Which plan has least cost is clearly very sensitive to the expected number of employees who have low salaries and the expected number who have high ages. Hence, data managers accumulate statistics from which such estimates can be made. Current systems keep track of some combination of:

the number of tuples in each relation
the minimum and maximum key values for any indexed field

This research was sponsored by the National Science Foundation under Grant DMC-8504633 and by the Navy Electronics Systems Command under contract N00039-84-C-0039.

a histogram of the number of tuples within various key ranges

Work on optimizing the collection of statistics is reported in [PIAT84, X, Y].

In the case that a query involves one or more joins, the query optimizer must estimate the number of tuples that will appear in any intermediate join. For example, one might run the following three-way join among the EMP, DEPT and PROJECT relation to find a list of the names of employees along with their department and the project name that they work on.

retrieve (EMP.name, DEPT.dname, PROJECT.name)
where EMP.dept = DEPT.dname and EMP.name = PROJECT.ename

If the optimizer chooses to join EMP and DEPT first, then it must guess the expected size of the partial answer found from:

retrieve (EMP.name, DEPT.dname)
where EMP.dept = DEPT.dname)

Of course, the cost of the remaining computation, that of joining the partial result with the PROJECT relation. Similarly, the query optimizer must guess the size of any other 2-way joins that are constructed by a query plan which it investigates. The size of the join of two relations, R_i and R_j is computed as: (s_{ij}) for two relations, R_i and R_j defined as:

$$n_{ij} = s_{ij} \text{ x } (n_i \text{ x } n_j)$$

where
n_{ij} : number of tuples in the result of the above join
s_{ij} : Selectivity of a join between relation R_i and R_j
n_i : number of tuples in relation R_i
n_j : number of tuples in relation R_j
Of course, n_{ij} must be estimated, and current optimizers (e.g. [SELI79]) utilize very crude techniques.

Whenever the actual selectivity differs from the estimated value, then the size of the actual intermediate relation will differ from its estimated value and the actual cost of the corresponding query plan will differ from its estimated value. Consequently, selectivity errors can lead to choosing a plan with minimum estimated cost but whose actual cost may be inferior to the actual cost of other possible plans. The purpose of this paper is to study the relationship between inaccuracies in estimates of join selectivities and the actual quality of the minimum cost plan selected using inaccurate statistics. This actual quality is found by comparing the selected plan with the actual optimal plan. In general, we have found that plan quality is very insensitive to inaccurate join selectivities. Hence, there is little reason to spend a lot of effort in constructing accurate statistics in this area. This result should be contrasted with [EPST80] where a different conclusion was reached for a distributed data base environment.

The remainder of this paper is organized as follows. In Section 2, we describe a simulated query planner similar to the one in [SELI79] which was our experimental tool. Section 3 turns to the design of our experiments and introduces the concept of a Q-factor which can be used to measure the sensitivity of a join order to variations in selectivity. Then, in Section 4 the results of our experiments are presented and analyzed in detail.

2. THE QUERY PLANNER

To perform this study we implemented a query planner which which accepts 4-variable and 5-variable queries and generates the optimal plan for executing them. The planner uses the same heuristics to determine the optimal plan as the System R planner described in [SELI79]. This

section discusses the assumptions made by our planner.

First, our planner accepts only join queries and assumes that select and project operations have already been performed. This allows us to concentrate on join selectivities exclusively.

The planner accepts the following inputs:
1) Number of relations involved
2) Cardinality of each relation
3) Tuple size of each relation
4) The relations which are sorted and the field on which they are sorted
5) The relations having secondary indexes and the fields on which they exist
6) Number of joins
7) The relations involved in each join and the joining fields from each relation
8) The selectivity of each join

Based on the above inputs the planner generates a plan that gives the following:
1) The sequence in which the joins should be performed
2) The kind of join to be performed -- merge-sort join or nested loop join
3) Whether any secondary indexes are used

Our query planner accepts both chain queries and non-chain queries. An example 4-variable chain query is the following:

R1 JOIN1 R2 JOIN2 R3 JOIN3 R4

An example non-chain query is:

R1 JOIN1 R2 AND R1 JOIN2 R3 AND R1 JOIN3 R4

Both chain and non-chain queries may be represented graphically as shown in Figure 1. The join clauses in in a query containing n joins are numbered:

c_1,c_2,.....,c_n

For example, the chain query noted above has three join clauses, c_1, c_2, and c_3.

The other major features of the planner are listed below.

The planner works by enumerating certain possible join sequences (also called nesting orders). As in [SELI79], we do not consider any join sequences whose adjacent relations in the sequence do not have a join clause present in the query. Such join orders are sure to be suboptimal. Hence, the number of join sequences considered is the number of permutations of the join clauses. A query with n joins is executed by performing n 2-way joins -- J_1, J_2, ...J_i,.....J_n. The cost of a 2 way join is calculated as follows:

$$CJ_i = (\#\ of\ \ disk\ \ I/O's)_i + \text{M x } (size\ of\ \ answer)_i$$

where

CJ_i : cost of performing the i^{th} 2-way join, J_i
$(\#\ of\ \ disk\ \ I/O's)_i$: number of disk I/O's required for join J_i
$(size\ of\ \ answer)_i$: size of the answer of the i^{th} join, J_i

The first term in the expression above represents the disk cost of processing the query while the expected size of the answer represents a measure of the CPU cost. The constant M is used to relate the different kinds of costs, and in our experiments we set M to 0.5. The total cost, CJ of a plan is the sum of the costs of the individual joins as follows:

$$\text{CJ} = \sum_i CJ_i$$

The planner considers two types of joins, nested loop joins and merge sort joins ([SELI79]). The nested loop join is performed by accessing all tuples of one relation (the outer relation, R_O) and matching each tuple against all the tuples of the inner relation, R_I. A fixed number of buffer pool

pages are assigned to each relation. The correct number of pages from the outer relation are brought into memory. The tuples on each of these pages is compared against all the tuples of R_I which are loaded into the buffer in groups of pages. The pages allocated to the inner realtion are then replaced by a new collection, and the above process is repeated. Thus, each tuple of R_O is brought into the buffer just once, while the tuples of R_I are brought in repeatedly till the join is completed.

In case a secondary index exists on the joining field of the inner relation, the alternative of accessing the tuples of the inner relation via this index is also considered. Performing a merge-sort join on already sorted relations requires loading each relation into iteratively into memory and then writing the result back. In case one or both relations are not already sorted, the cost of sorting the relations is added in determining the cost of the join. For every join sequence being considered, the planner computes the cost of the individual joins using the cheapest method and adds up the cost of all joins in the sequence to determine the total cost. After computing the cost for each join order, the one with the lowest cost is selected as the optimal plan. We assume that all the join clauses are independent. For example, in the chain query above, if the join between R1 and R2 was done first, the next join between R12 and R3 would have the same selectivity as the join between the original R2 and R3. The page size was set to 4096 bytes. Secondary indexes are implemented as B-trees. In order to determine the number of levels within a B-tree index, we assume an index page contains 100 entries. Hence a 10^6 tuple relation requires a 3-level index while a 10^7-tuple relation would require a 4-level index. If the tuples of the inner relation are accessed via an index while performing a nested loop join, the root page of the index is loaded into memory once and kept in the buffer for subsequent use while the lower-level index pages are fetched from disk every time a new tuple has to be accessed.

3. EXPERIMENTAL SETUP

3.1. Design of Experiments

The purpose of the experiments was to investigate the effect of varying selectivities on the optimal nesting order. We have performed several experiments on two collections of 4-variable and 5-variable queries. Figure 2 gives a list of 10 4-variable queries that we chose for the experiments. Figure 3 gives a list of 10 5-variable queries while Figure 4 gives the sizes of the relations considered. Each relation is assumed to be 100 bytes wide. In each group we have included an equal number of chain and non-chain queries. They represent a good cross-section of queries that are likely to arise in practice. Each group is composed of queries with varying number of secondary indexes and relations that are sorted on the joining fields. Hence, cases optimizable by both merge-sort and nested loops iteration are included.

In all of our experiments we have varied the selectivities of one or more join clauses over a range of values and determined the corresponding optimal nesting order. The selectivity values were varied over a range of 0.33 to 3 times an initial value for all our experiments. This corresponds to an order of magnitude difference between the smallest and the largest selectivity values used in a range. The initial values of selectivities were chosen such that the size of the result of a multi-way join was of the order of the size of the smallest relation in the join. We have observed that a very large number of real life queries conform to this rule. Thus, in a two way join between R1 having 10^3 tuples and R2 having 10^5 tuples, we would expect an answer of 10^3 tuples. This rule served as a guideline in determining the initial selectivity values which are also indicated in Figure 4.

Our experiments varied the important model parameters over wide ranges. Before discussing details of the experiments we introduce the concept of Q-factor which is our metric for assessing the sensitivity of a nesting order.

3.2. Q-factor

The optimal nesting order of a query is the minimum cost sequence of joins. The performance of any nesting order for a combination of selectivities can be judged by the following ratio:

$$r(i,j) = c(i,j) / c(i_{opt},j)$$

where

r(i,j) : ratio of the cost of nesting order i to the cost of the optimal nesting order, i_{opt}, for selectivity combination j

c(i,j) : cost of nesting order i for selectivity combination j

By definition a value of r(i,j) close to 1 indicates a good nesting order. A nesting order which is optimal for one set of selectivity values may not continue to be optimal as the selectivity values are varied. We define Hence, we define the quality factor (or Q-factor), Q for a query as:

$$Q = \min_{i} \left(\underset{j \in J}{avg}\ r(i,j) \right)$$

where J : set of selectivity values j in the range over which selectivity is varied

The value i for which the above expression is minimized is the best nesting order, i_{best}. Therfore, the best nesting order has the minimum average cost ratio r(i,j) over the range of selectivity values among all nesting orders. Q is our metric for defining sensitivity of a query to variations in selectivity. Hence we call it the Quality factor for a query. The minimum value of Q is 1, and a value of Q close to 1 means that the query is insensitive to variations in selectivity. Thus, the cost of processing the query using the best nesting order, i_{best} is likely to be very close to the optimal cost at all points over the range J. A higher value of Q denotes greater sensitivity to variations in selectivity and means that the cost of processing the query by the best nesting order is likely to be considerably sub-optimal for selectivity values over the range J.

A second definition for Quality factor, Q1 is the following:

$$Q1 = \min_{i} \left(\underset{j \in J}{max}\ r(i,j) \right)$$

Again, the value of i for which the above expression is at its minimum is the best nesting order, i_{best}.

The difference between the two definitions for Quality factor is that the first chooses the nesting order with the best average performance over the range of selectivity values while the second selects the one with the best worst-case performance. In general, the best nesting order determined by using the alternative definitions of Quality factor will not always be the same. However, it turns out that in most cases both the metrics lead to the same best nesting order.

We use Q as our criteria for selecting the best nesting order in all our experiments. However, in the first set of experiments in Section 4.1 we also illustrate how Q1 is determined.

4. RESULTS OF EXPERIMENTS

4.1. introduction

Four sets of experiments were carried out on a set of 20 queries. Important parameters like initial selectivities and relation sizes were varied to study the effect under changing parameter values. In the first set of experiments described in Section 4.2, we varied the selectivity of one join clause over a wide range while the selectivity of another join clause was varied in inverse

proportion so as to keep the size of the answer fixed. Section 4.3 discusses experiments in which the selectivity of only one clause was varied over the same range as before and the size of the answer was allowed to vary in the same proportion. In the third set of experiments reported in Section 4.4, the size of the answer was allowed to increase in intervals by increasing each selectivity by a constant factor in the same proportion. Section 4.5 describes experiments in which the size of one of the relations was allowed to vary over a wide range while a corresponding selectivity value was also varied so as to keep the size of the result equal to the size of the smallest relation. Finally section 4.6 summarizes the conclusions that can be drawn from the four sets of experiments.

4.2. Fixed Answer Size

This experiment was conducted on the set of 4-variable and 5-variable queries explained in the previous section. We varied the selectivity of clause c_1 from 0.33 of its initial value to 3 times the initial value. The selectivity of clause c_3 was changed in inverse proportion in order to keep the size of the answer fixed. At 8 different points over the selectivity range, we determined the cost of executing each query by the alternative join orders discussed in Section 2. For each query we determined the best nesting order, i_{best}, over the range and computed the ratio r(i_{best},j) at the 8 points over the range. The r(i_{best},j) values for the 4-variable and 5-variable queries are tabulated in Tables 1 and 2 respectively. Each row lists the r(i_{best},j) for a query at the 8 different selectivity values. The r(i_{best},j) values are used to compute Q and Q1 for each query. Q is calculated as the average of the 8 r(i_{best},j) values in each row, while Q1 is the maximum of the 8 values. The last row in Tables 1 and 2 gives the average values for the 10 queries in each set. It represents the expected performance of a query picked at random from our collection of queries. In Figures 5 and 6 we have plotted the r(i_{best},j) ratios for the average query, and for the queries with the best and worst behaviour.

The Q-factor for all 4-variable queries is less than 1.10 which clearly illustrates that on the average, the cost of the best nesting order is within 10% of the cost of the optimal over the entire range of selectivity values. The Q1 values in the last column of Table 1 show that in only 2 of the 10 queries (queries 2 and 3), the worst-case cost of the best nesting order is 50% more than the cost of the optimal nesting order. For the remaining 8 queries the worst-case cost of the best nesting order is considerably less.

The results of the experiments with 5-variable queries (Table 2) confirm the above observations. For 9 out of 10 queries, the value of Q is within 1.10 again, indicating a less than 10% sub-optimality in the cost of the best nesting order on the average. For 1 query (query 6) the value of Q is 1.47. The value of Q1 is within 1.50 for 9 out of 10 5-variable queries, indicating that in the worst case, there is a less than 50% sub-optimality in the cost of the best nesting order.

The above experiments show that even when selectivities are varied over a wide range, the deterioration in the cost of the best nesting order relative to the cost of the optimal is minimal for most queries. Therefore, if a query has a low Q-factor, it is certainly a good idea to execute it using the best nesting order.

4.3. Varying One Selectivity

The experiments of the previous section were modified slightly by allowing the selectivity of only one join clause to vary. We varied the selectivity of the first clause, c_1 over the same range as before. The rest of the parameters were the same as in the previous section. The experiments of the previous section were then repeated for the 4-variable and 5-variable queries. Notice that in this case the size of the answer was allowed to vary as the selectivity of the first clause was

changed. We obtained the same data as before and used it to extract the Q-factor for each query. Tables 3 and 4 give the Q-factors for the 4-variable and 5-variable queries, respectively.

The second set of experiments reinforce the results of the first set. 7 of the 10 4-variable queries have a Q-factor of 1.00. The remaining 3 queries have a Q-factor of less than 1.10. 9 out of our 10 5-variable queries from Table 4 have a Q-factor within 1.06. Query 6 has a Q-factor of 1.20.

4.4. Incrementing The Answer Size

For the experiments discussed above, the size of the answer was of the same order of magnitude as the smallest relation in the join. In this section we scaled all selectivities by a multiplier in order to increase the size of the answer. Starting with an answer of 1000 tuples, we let the answer size of our sample queries increase up to 10^6 in multiples of 10. Since the selectivity of each clause is increased by the same multiplier the relative selectivities of the various join clauses do not change.

Tables 5 and 6 show the results for 4-variable and 5-variable queries, respectively. For each query the tables give the values of Q-factor as the size of the answer varies over the range from 10^3 to 10^6. The last row in each table gives the average Q-factor values for the 10 queries corresponding to a certain number of tuples in the answer.

Even with increasing size of the answer, there is minimal deterioration in the Q-factor values. The largest Q-factor observed in Table 5 is 1.10 (query 4, 10^6 tuples). The largest average value across all 4-variable queries (see last row of Table 5) is 1.03. Among the 5-variable queries the average value of Q-factor across all queries (see last row of Table 6) stays within 1.10. The highest Q-factor in Table 6 is 1.47 (query 6, 10^3 tuples).

We observe that even though some queries (for example, queries 7 and 10 in Table 6) exhibit a Q-factor that increases with the answer size, the pattern is not maintained in the average values.

4.5. Varying The Size of One Relation

The final set of experiments reported here were done to study the effect of changing relative relation sizes and relative selectivities. We allowed the size of one of the relations to vary over a wide range and also allowed one of the selectivities to change so that the size of the result was maintained constant. In order to study the effect on highly sensitive queries we selected two queries with the highest Q-factor values from each of Tables 1 and 2. For each query we increased the size of the second relation R2 from 100 tuples to 10^7 in multiples of 10. We adjusted the selectivities in such a way so as to get an answer equal to the size of the smallest relation. Tables 7 and 8 show the Q-factor values with varying R2 sizes for 4-variable and 5-variable queries respectively.

The experiments of this sub-section were intentionally designed to represent a worst case because (some good reason). The highest Q-factor values for the 4-variable queries in Table 7 are 1.15 (for both queries 2 and 7). Other Q-factor values in the table are substantially lower. The highest Q-factor values for the 5-variable queries in Table 8 are 1.22 (query 2) and 1.48 (query 6).

4.6. Analysis of The Results

In the four sets of experiments above, we have varied several important parameters for 20 different queries in order to get representative results. We have also identified the 4 most sensitive among the 20 queries and presented worst-case results for them. Among all the 20 queries,

there was only one query (query 6 in Figure 3) which exhibited markedly greater sensitivity to changes in selectivity compared to others. It is the only query in which it is conceivable that better apriori estimation of selectivity could result in reasonable savings. For the remaining 19 out of 20 queries, it is highly unlikely that savings in execution cost would offset the additional cost incurred by constructing accurate estimates of selectivities.

Our results show an excellent average behaviour of best nesting orders both for 4-variable and 5-variable queries. We find that when the size of the result is of the same order as the size of the smallest relation, most queries exhibit low sensitivity to variations in selectivity as indicated by the the Q-factor values. Even when the size of the result was increased by multiplying all selectivities by a constant factor, the Q-factor values did not deteriorate.

These results suggest a simple scheme for optimizing multi-join queries. One should simply choose a "middle of the road" value for each join selectivity. This crude estimate is likely to yield good performance. A more sophisticates scheme would be to vary the selectivity about an approximate value. This will allow the determination of the best nesting order and the construction of the corresponding Q-factor. If the Q-factor is within a certain value, like 1.20, the best nesting order should be chosen for processing the query. Our experiments show that this will be true for a very large number queries that arise in practical situations. A more accurate determination of selectivity using detailed statistics (e.g.,[KOOI82],[PIAT84]) is warranted only for those queries for which no insensitive plans can be found.

5. CONCLUSIONS

This paper has presented a study of the sensitivity of query plans to errors in the estimates for join selectivities. Since the cost of making exact determination of selectivities is prohibitive, estimates of selectivity are at best approximate. Current query planners take these estimates as exact and determine the optimal processing plan based on them. to study sensitivity of query plans to selectivity errors, we introduced a Q-factor for measuring sensitivity of queries to variations in selectivity. The value of this metric was measured for 20 different queries under different parameter values. Surprisingly low Q values were observed, and we can conclude that data managers should make crude selectivity estimates and not spend a great deal of time attempting to provide accurate selectivity values.

REFERENCES

[DATE84] Date, C.J., A Guide to DB2, Addison-Wesley Publishing Co., 1986.

[DATE86] Date, C.J., An Introduction to Database Systems, Vol I, Addison-Wesley Publishing Co., 1986.

[KOOI82] Kooi, R. and Frankfurth, D., "Query Optimization in INGRES," IEEE Database Engineering, September 1982.

[PIAT84] Piatetsky-Shapiro, G. and Connell, C., "Accurate Estimation of the Number of Tuples Satisfying a Condition," Proc. 1984 ACM-SIGMOD Conference on Management of Data, Boston, Mass. June 1984.

[RTI84] Relational Technology, Inc., "INGRES Reference Manual, Version 3.0," November 1984.

[SELI79] Selinger, P. et. al., "Access Path Selection in a Relational Database Management System," Proc. 1979 ACM-SIGMOD Conference on Management of Data, Boston, Mass., June 1979.

[WONG76] Wong, E., Youssefi, K., "Decomposition - a Strategy for Query Processing," ACM TODS, 1,3 (Sept 1976).