An. Şt. Univ. Ovidius Constanţa Vol. 13(2), 2005, 75–82

Cost evaluation in joins' optimization

Nicoleta Liviana Tudor

Abstract

This article presents an approach of the cost model used in join optimization. The search space is determined through transformations on the query blocks, depending on the selection predicate. Different implementations of the JOIN operator are taken into account for cost evaluation of the execution plans.

Subject Classification: 168P15.

1 Introduction

A method to optimize the access at the objects of a relational database is through the optimization of the queries, witch can be heuristical or systematical.

In systematic optimization, the cost of the execution of one query is systematically decreased through the estimation of the costs for various execution plans. The cost concerns the I/O operations (hard disk acces, intermediary results storage), the CPU operations (memory transfer) but also network data transfer for client/server databases. The cost functions have as arguments some metadata that are stocked in the catalog system of the database as well as the blocking factor, the number of levels of a multilevel index, the number of blocks of the first level index, the number of distinct values of an indexation attribute and the cardinality of a selection.

This article suggests an approach of the cost model used in joins' optimization.

The search space associated with the JOIN operator is determined through transformations that are applied to the join's blocks and the cost of the execution plans is evaluated, the one with the lowest cost being chosen. So, for a given query, there are two problems that arise: witch is the search space and how to implement the search.

Key Words: Search space; Block transformation; Cost functions; Query tree.

2 The search space for the JOIN operator

The relational algebra JOIN operator, used for composing relations, is implemented in SQL language as follows:

```
SELECT  c1 [, c2, ]  FROM R, S[, T] WHERE p [^| v q ...]
```

where: c_i, $i \in \{1, 2, \ldots\}$ represents attributes or expressions with attributes; R, S, T - the relations from wich data are extracted; p, q - predicates that must have the truth value "*true*" and are applied to the tuples.

For the search space determination, a query is decomposed in blocks that are independently represented and optimized.

For a query block, the equivalent transformations are executed, the physical operators are chosen, and the existent indexes are used or other indexes for sorting are built.

For the JOIN operator ($\triangleright\triangleleft$), the transformations of the blocks for the realization of set operations are:

$$R \triangleright\triangleleft S = S \triangleright\triangleleft R$$

$$(R \triangleright\triangleleft S) \triangleright\triangleleft T = R \triangleright\triangleleft (S \triangleright\triangleleft T)$$

The transitivity can be represented using trees as follows (figure 1):

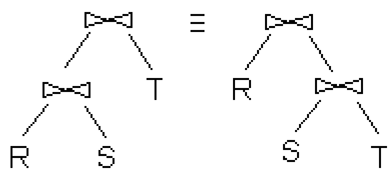


Figure 1: Tree representation of the transitivity.

When the join operation implies a selection (σ), and the logical predicate is complex, being a conjunction/ disjunction of predicates (let's assume that the p predicate contains only attributes of the relation R and that the predicate q uses only attributes of the relation S), the transformations of the query's blocks [1] can be:

$$\sigma_p(R \triangleright\triangleleft S) \equiv [\sigma_p(R)] \triangleright\triangleleft S$$

$$\sigma_q(R \triangleright\triangleleft S) \equiv R \triangleright\triangleleft [\sigma_q(S)]$$

If the join operation also contains a projection (π), and x is a subset of attributes of R, y is a subset of attributes from the relation S and z is the intersection of the attributes from R and from S that are used by the p predicate, then the transformations of the JOIN blocks can be:

$$\pi_{xy}\left[\sigma_p(R \rhd \lhd S)\right] = \pi_{xy}\{\sigma_p\left[\pi_{xz}(R) \rhd \lhd \pi_{zy}(S)\right]\}$$

The search space can be represented as in figure 2:

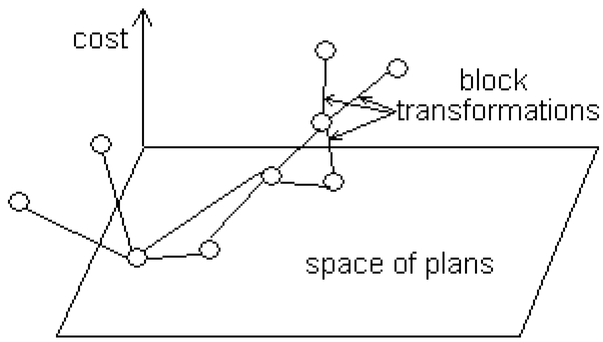


Figure 2: The search space.

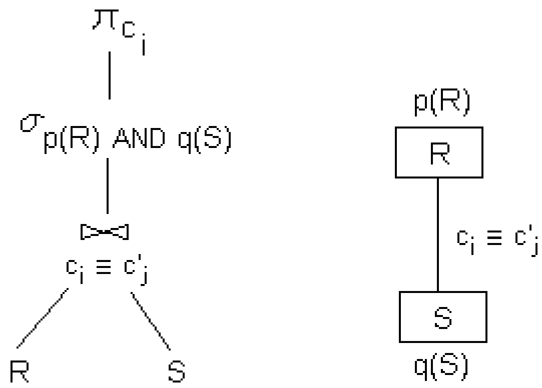
3 The representation of a join

In order to represent a join operation, the tree of the query is used. The nodes of this tree are the operators resulted from decomposing into elementary operations and simple predicates; the terminal nodes (the leafs) represent the relations that are composed.

For exemplification, let us consider two relations with the relational schemas $R(c_1, c_2, \ldots, c_n)$ and $S(c'_1, c'_2, \ldots, c'_m)$ and an implementation of the JOIN operator that is based on a conjuction of two predicates:

```
SELECT R.ci FROM R, S WHERE R.ci = S.c'j AND p(R) AND q(S),
```

where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $c_i \equiv c'_j$, p(R) is a predicate with attributes from R and q(s) is a predicate with attributes from S. The join operation uses the projection on the c_i attribute of the relation R. So, the tree associated to this join operation can be represented as in figure 3.a and the query graph is represented in figure 3.b:



a) b)

Figure 3: a) Query tree, b) Query graph.

Through the sorting of the operations in the query tree and the choice of an algorithm of search in the planes' space, various execution plans are obtained and the one with the minimal cost is chosen.

4 Evaluation of costs

Let R and S be two relations with the corresponding relational schemas $R(c_1, c_2, \ldots, c_n)$, $S(c'_1, c'_2, \ldots, c'_n)$ and the equi - junction of the relations R and S:

```
JOIN (R, S; R.ci = S.c'j),
```

where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $c_i \equiv c'_j$.

Let's note s_j - the selectivity of the join operation (the percent of records that satisfy the condition of the join, being computed as a ration between the number of tuples of the join and the number of tuples of the cartesian product), $|R|$ or $|S|$ - the number of records of the R relation, and S relation respectively, bR - the number of blocks from the R relation, bS - the number of blocks of the S relation, $f_{bloc}R$ - the factor of blocking of the table result. Then:

$$s_j = \frac{|R|X|_c S|}{|RXS|} = \frac{|R|X|_c S|}{|R| * |S|},$$

where c is the predicate R.ci = S.c'j.

If the ci attribute is a key of the R relation, then $|R|X|_c S| <= |S| \rightarrow s_j <= 1/|R|$. If the c'j attribute is a key of the S relation, then $|R|X|_c S| <= |R| \rightarrow s_j <= 1/|S|$. So $0 <= s_j <= 1$.

In order to estimate the size of the join file, different implementations of the JOIN operator are taken into account.

The implementation Nested loops of the JOIN operator uses iterative imbricate structures:

```
for r in R
for s in S
if r. ci = s. c'j then
select (r,s)
endif
repeat
repeat
```

The cost function for the processing of these instructions is:

$$C = b_R + (b_R * b_S) + \frac{s_j * |R| * |S|}{f_{bloc}R},$$

where $b_R * b_S$ is the cost of the execution of the two iterative structures for' and $(s_j * |R| * |S|)/f_{bloc}R$ is the cost of storing the results.

The JOIN operator can use an index for the c'j attribute of the S relation, the algorithm being:

```
for r in R
uses index structure from S
get all s from S for s. c'j = r. ci
repeat
```

The cost function when using an index for the c'j attribute (the S relation) is:

$$C = b_R + |R| * (x + cs) + \frac{s_j * |R| * |S|}{f_{bloc}R},$$

where x is the number of levels of a multilevel index and cs is the cardinality of the selection (the average number of records that satisfy the equality of the selection - for the key: cs = 1; for the attributes that are not keys: cs is the number of records of the relation / the number of distinct values of an indexation attribute).

The implementation of the JOIN operator that is based on sorting on the same attributes (the R table is sorted on the c_i attribute and the S table on the c'_j attribute) can be algorithmically expressed as follows:

```
i = 0; j = 0
while i<=nR and j<=nS
// nR (nS respectively) = number of tuples from R (S respectively)
```

```
if R(i). ci < S(j). c'j then
i=i+1
else
   if R(i). ci > S(j). c'j then
j=j+1
else
select tuples (R(i), S(j))
makes other tuples with R(i) increasing j
creates other tuples with S(j) increasing i
endif
endif
repeat
```

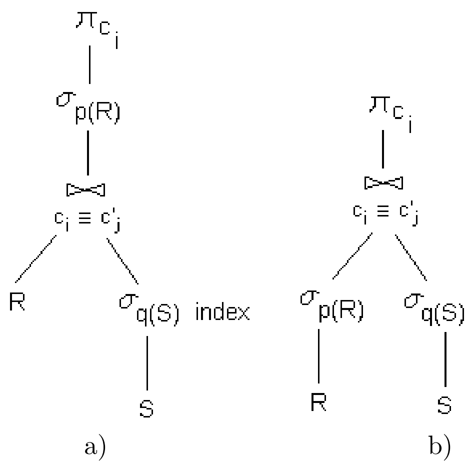


Figure 4: a) Plan with index, b) SORT - JOIN.

The cost function in the case of sorting is:

$$C = b_R + b + S + \frac{s_j * |R| * |S|}{f_{bloc}R}.$$

If an index is used for an attribute from S, then the execution plan of the join operation that is used as example in the precedent paragraph can be represented as depicted in figure 4a. For the implementation that is based on sorting, the join plan is presented in figure 4b.

5 Conclusions

This article presents an approach of the cost model used in join operation optimizations. The search space is determined through transformations on the query blocks, depending on the selection predicate. The generation of the execution plan of queries permits the interpretation of the statistics and estimated costs. References

References

[1] Ramakrishnan, Gehrke, Molina Garcia, Chaudhuri Surajit, *Query Optimization*, Technical report, Hewlett-Packard Laboratories, Palo Alto, 1995.

[2] Holowczak Richard, *Database Management Systems II*, Baruch College, New York, 2001.

[3] Chaudhuri Surajit, Shim Kyuseok, *An Overview of Cost-based Optimization of Queries with Aggregates*, In Proceedings of the 5th International Conference on Extending Database Technology, France, 1996.

Petroleum-Gas University of Ploiesti,
Department of Computer Science,
Romania
e-mail: tudorlivia@yahoo.com