Configurable and Extensible Query Optimization by Controlled Term Rewriting

Mazeyar E. Makoui
{mem}@dbs.uni-hannover.de
University of Hanover

Abstract

Classic query optimization in relational database systems relies on phases (algebraic, physical, cost-based) and heuristic strategies for these phases (e. g. push selections). This, however, proves to be too inflexible not only for certain standard situations, but in particular for non-standard, e. g. spatial or multimedia applications which introduce expensive selection and join predicates, and which could profit from computing redundant data like indexes during query execution. Our goal is a uniform development environment for query optimizers of object-relational DBMSs. Therefore, we propose to base optimization on controlled term rewriting. This framework uses a general spectrum of operators covering relational algebra operators, their physical implementation alternatives, non-standard predicates, etc. The application of rewriting rules between corresponding terms should be controlled a) locally by rule-specific conditions which can consider syntactical as well as quantitative (size/cost-dependent) criteria, b) compositionally by regular sequencing patterns, and c) globally by strategies for optimum searches. We have developed an optimizer simulator which is thus extensible wrt operators and configurable wrt control mechanisms.

1 Introduction

Classic optimization in relational database systems is divided into three phases (algebraic, physical and cost-based optimization) and is based on heuristic rules (at least as it is implemented in common query optimizers). But it is easy to find counter examples against heuristic rules already in classic relational databases, e. g. against the well-known recommendation that selections should be executed as early as possible. It is even more striking that non-standard operators in object-relational databases, in particular new selection predicates or new projection terms like geometric operators, which are drastically more expensive than conventional ones, cannot be handled by classic heuristics.

Our experiences [12] with spatial database programming show that it can even be worthwhile to generate indexes on intermediate data during query execution; this does not fit into the classic paradigm that indexes either exist - on base relations only - or do not exist before a query is executed.

Fortunately, the known methods of optimization can be extended: there are rewriting rules on relational algebra terms, physical implementations have been proposed as "executable" algebra operators, and cost-based optimization can handle extra investments like sorting or indexing a relation. The goal is a uniform development environment for query optimizers of object-relational DBMSs.

Therefore, we propose to base optimization on controlled term rewriting. First, it utilizes a very general underlying notion of operators covering: a) relational algebra operators, b) expensive predicates and functions from non-standard applications, c) physical implementation alternatives of those operators, d) auxiliary operators creating redundant or organized data like sorting, indexing or even materialized views. Second, application of rewriting rules should be controllable: 1) *locally* by specifying individual conditions under which a rule may be applied, i. e. conditions which consider syntactical (schema-dependent) restrictions, but which can also consider quantitative (size/cost-dependent) criteria 2) *compositionally* by specifying patterns describing the sequencing, iterating, or alternating between the rules which should be applied 3) and rather *globally* by general strategies, e. g. from heuristics for optimum searches. We have developed a query optimizer called `RELOpt` which is: i) extensible wrt to the operators ii) and configurable wrt to the control mechanisms.

This query optimizer serves as a vehicle for better understanding and validating conventional heuristics as well as non-standard extensions. Of course, we expect that elements from that simulator can be utilized for future optimizers that will need more systematic influencing by (advanced) users than current optimizers allow in order to adapt object-relational databases to the needs of non-standard applications. In particular, our optimizer offers the following capabilities: i) All possible operators can be considered with their various implementations. ii) All possible orders of the various operators can be considered. iii) Every possible (pre-defined) optimization strategy can be applied. iv) All optimization possibilities are considered in only one phase and not in separated phases. Last, but not least, our optimizer also offers control menus to limit exploration of thus vast search space.

2 Related Work

The research most comparable with our own was that carried out as part of a joint venture between the universities of Konstanz and Rostock between 1995 and 1999. The "Cost- and Rulebased Optimization of object-oriented QUEries" (CROQUE)-Project had the goal to develop an optimizer for an object-oriented query evaluation system with three characteristics: a) a descriptive, object oriented query language based on ODMG-OQL, b) logical data independence (i. e. separation of conceptual and physical level) and c) a cost- and rule-based optimizer translating a descriptive query of the conceptual level into a query execution plan dependent on the chosen storage structure.

Our focus is on the last item, which was first presented in 1996 [6]. A heuristical approach for ordering algebraic rewrite rules was published in 1998 [11]. The research about query rewriting and search in CROQUE was concluded in 1999 [10]. The main contribution of that project was to develop a more flexible order for algebraic rules by preferring heuristic optimization concepts. One can sum up the idea by using rules with more "optimization potential" over using rules which do not deliver cost improvements. As a result one obtains a set of algebraic rules which might be better for further optimization than other ones.

Since this approach is limited to algebraic rules there is no possibility to create indexes or sortings. The reason is that one would violate the separation of logical and physical level, which however leads to an explosion of rules and possibilities (e. g. ,the COKO-KOLA optimizer [2] (1998) consists of 60 operators and altogether about 600 rules).

Rule ordering was first presented by EXODUS [3] (1987) and its successor Volcano [4]. Even though the optimizers could change the ordering in a limited way, they did not give the user a possibility to adjust it to their own data pool and operators.

Programming the rule ordering was proposed by COKO-KOLA [2] (1998), Starburst [5],[13] (1989/92) and GOM [9] (1990), which also limits the optimization potential and thus lacks flexibility, e. g. for developing new strategies for object-relational operators.

Calculating costs for all variants of query execution including the original one was presented in EXODUS [3] (1987) and Gral [1] (1992). This goal was achieved by a very expensive way of calculating the cost of every query execution plan or subplan. We suggest to precalculate costs of all subplans in order to efficiently recalculate costs for alternative query execution.

The most recent work was done in the OPT++-Project [8],[7] (1999/2000), by basing the optimization procedure on an object-oriented framework. Even though this approach has an architecture that significantly improves the extensibility and maintainability of a query optimizer, it is still restricted to algebraic and physical phases, but does not consider cost-conditions.

To compare our contribution to well-known approaches of the last years, we use a classification which was introduced in [11] and enhance it with one reasonable last question: 1) Does the system use a rule selecting mode (e. g. a rule file)? 2) Does the system offer a conditioned, on- resp. offline or hard coded rule ordering? 3) What kind of dependencies can be used for rule ordering? a) cost-based b) statistics-based c) heuristic-based 4) Does the system combine different heuristic search strategies? 5) Is extensibility for rules hard coded or is a flexible language used? 6) Does the system consider object-relational operators?

System criterion	Volcano	EXODUS	COKO-KOLA	Starburst	GOM	Gral	CROQUE	OPT++	RELOpt
1)	yes	no	no	yes	yes	yes	yes	yes	yes
2)	online	online	prog.	prog./ online	prog.	online	none	prog.	prog./ online/ cond.
3a)	yes	yes	no	no	no	yes	yes	yes	yes
3b)	yes	yes	no	no	no	no	yes	yes	yes
3c)	yes	no	yes	yes	yes	yes	yes	yes	yes
4)	no	no	no	no	no	no	no	yes	yes
5)	prog.	no prog.	prog.	prog.	prog.	no prog.	no prog.	prog.	prog./ lang.
6)	no	no	no	no	no	no	no	no	yes

3 Conditional term rewriting

The idea of conditional term rewriting is no longer to apply term rewriting heuristically, but to make it dependent on cost functions. By this, we enlarge the traditional algebraic term rewriting rules by cost conditions.

As a basic element to specify query optimization, we utilize conditional term rewriting rules:

$$\tau_1 \underset{2}{\overset{1}{\rightleftharpoons}} \tau_2 \quad 1) \text{ if } \langle \mathsf{conditions}_1 \rangle \quad 2) \text{ if } \langle \mathsf{conditions}_2 \rangle$$

which generalize rewriting of relational algebra terms and whose conditions can consider well-known syntactical as well as unconventional quantitative criteria.

Rewriting rules are composed of relational algebra operators, e. g. selections, projections and joins:

$$(\mathsf{R}_1 \bowtie_{\varphi_1} \mathsf{R}_2) \bowtie_{\varphi_2} \mathsf{R}_3 \rightleftharpoons \mathsf{R}_1 \bowtie_{\varphi_1} (\mathsf{R}_2 \bowtie_{\varphi_2} \mathsf{R}_3)$$

Derived from these base operators we enrich our system with their implementation variants (physical operators) and corresponding conditions, e. g.:

$$(\mathsf{R}_1 \bowtie_{\varphi_1} \mathsf{R}_2) \bowtie_{\varphi_2} \mathsf{R}_3 \overset{1}{\rightleftharpoons} \mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel,\ Rel}} (\mathsf{R}_2 \bowtie_{\varphi_2}^{\mathsf{Rel,\ Index}} \mathsf{R}_3)$$

$$1) \text{ if } \mathsf{index\_supported}(\varphi_2, \mathsf{R}_3)$$

Rel stands for "relational scan", Index for an "index scan". Rel, Rel thus denotes the nested-loop join. Rel, Index the variant with an outer relational scan and an index-based access to the inner relation. Of course, there must be an appropriate index support as stated in the condition.

To every rule, two kinds of conditions may be added: 1) syntactical conditions using metadata like $\mathsf{attributes}(\varphi)$, $\mathsf{schema}(\mathsf{R})$, $\mathsf{sorting}(\mathsf{R})$ or $\mathsf{index\_supported}(\varphi, \mathsf{R})$ and 2) quantitative conditions using statistical or computed data like $\mathsf{size}(\mathsf{R}) = |\mathsf{R}|$, $\mathsf{cost}(\tau)$, $\mathsf{selectivity}(\varphi, \mathsf{R})$, etc.

An example rule could be:

$$\left(\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel,Rel}} \mathsf{R}_2\right) \bowtie_{\varphi_2}^{\mathsf{Rel,Rel}} \mathsf{R}_3 \overset{1}{\rightleftharpoons} \mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel,Rel}} \left(\mathsf{R}_2 \bowtie_{\varphi_2}^{\mathsf{Rel,Index}} \mathsf{R}_3\right)$$

$$1) \text{ if } \mathsf{index\_supported}\,(\varphi_2, \mathsf{R}_3) \ \wedge \mathsf{cost}\left(\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel,Rel}} \mathsf{R}_2\right) \geq \mathsf{cost}\left(\mathsf{R}_2 \bowtie_{\varphi_2}^{\mathsf{Rel,Index}} \mathsf{R}_3\right)$$

Moreover, we introduce operators which can create redundant or organized data like sorting, index creation (on the fly) and even materialized views, e. g. when using a merge-join ($\bowtie_{\varphi}^{\mathsf{Sort,\ Sort}}$):

$$\mathsf{R}_1 \bowtie_{\varphi} \mathsf{R}_2 \overset{1}{\rightleftharpoons} \mathsf{sort}(\mathsf{R}_1) \bowtie_{\varphi}^{\mathsf{Sort,\ Sort}} \mathsf{R}_2$$

$$1) \text{ if } \mathsf{sorting}(\mathsf{R}_2) \cap \mathsf{attributes}(\varphi) \neq \emptyset$$

The function $\mathsf{sorting}(\mathsf{R})$ gives the attribute sequence by which R was ordered.

Naturally, such operators may temporarily increase costs in order to save them later on. Therefore, a the cost control must be weakened to a certain extent (compare section 5).

For implementation purposes we assume that terms are represented by attributed trees where every node carries all costs and size results for its subtree. This procedure saves complete recalculation for new variants.

A rewrite rule is applied to a term by substituting each occurrence of the left-hand side by the right-hand side and thus generating a new variants for every substitution. If the rule is applicable, then generated variants are stored for further optimization in a hash-indexed table. To get variants with multiple substitutions, the rule must be iterated (see section 4).

4 The Compositional Control System

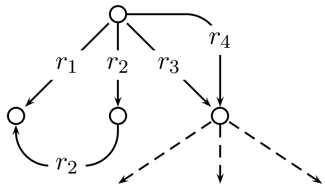
The order in which rules may be applied, can be controlled by following regular sequencing control: 1) Alternatives can be written with the help of a vertical bar which defines the parallel application, e. g. $\mathsf{r}_1|\mathsf{r}_2$. 2) Groups are defined by parentheses which scope and prefer the rules, e. g. $(\mathsf{r}_1|\mathsf{r}_2)|\mathsf{r}_3$. 3) Quantifiers after a rule or group specify how often that preceding expression is allowed to occur: a) an exclamation mark '!' indicates that at least 1 of the previous rules must have been applied; b) the asterisk '*' indicates that the previous rules must be applied as often as possible. c) A question mark '?' indicates that there is 0 or 1 of the previous rule applied. In addition to that we need constructs that help to terminate the generation of variants: 4) By using square brackets one can "evaluate" the generated variants, e. g. for choosing the 100 best variants we write $[(\mathsf{r}_1|\mathsf{r}_2|\mathsf{r}_3)^*]_{100}$. For returning the optimized execution plan we finally use $[...]_1$. With this notation we are also able to specify phase-leaned optimization, e. g.

$$[(\langle\mathsf{algebraic\ rules}\rangle)^*]_1[(\langle\mathsf{physical\ rules}\rangle)^1]_1.$$

For example if we would like to optimize by a brute-force strategy (executing in each iteration every existing rule in parallel), we would write $[(\mathsf{r}_1|\mathsf{r}_2|\mathsf{r}_3|\mathsf{r}_4)^*]_1$.

This is done as long as new variants, which differ from the known ones are produced. Repeated generation of the same variants is detected by the hash-indexed table.

One can clearly see that this procedure is not effective because of its exponential growth. Therefore, it is necessary to take into consideration pruning methods which help us to deliver good results by comparing less subsets.



5 Global Control System

Starting with a direct transformation from a relational operator tree we do have an initialization point (an estimated execution cost) for the transformation. Additionally, we have a set of conditioned transformation rules which are derived from algebraic transformation rules by physical operators and enriched with several conditions like cost-conditions. In general, all of the rules must be matched against every operator node in the given general tree to find out the applicable rules needed in order to optimize the given execution plan.

In our case this matching does not take place with all the rules, since only a subset of the global rule set must be matched. This subset is limited by conditions which prevent senseless matching, and by the user who can input his knowledge about the data pool. Such knowledge could be the lack of special kinds of object-relational operators, which therefore do not have to be matched anyway. This decreases ”exhaustive matching” right at the beginning of the optimization. Therefore, an enriched rules-order strategy is one of the most important parts in our optimization strategy.

The following presented strategies give us the possibility to narrow the search space in different ways.

5.1 Static Cost-Threshold

As a native first approach we could consider variants that have equal or less cost results than their predecessors. Therefore, it is very important that the starting point is not in a local minimum which cannot be improved.

Such Hill Climbing strategy is more flexible by using a small buffer called s. In this case it is senseless to invest in sorting or index building, because it always exceeds the cost limitation. But on the other hand, we will have a good local minimum after a short time of search.

$$\tau_1 \underset{}{\overset{1}{\rightleftharpoons}} \tau_2$$

1) if $\mathsf{cost}(\tau_2) \leq \mathsf{cost}(\tau_1) + \mathsf{s}$

$\mathsf{s} = 0$ delivers Hill Climbing

5.2 Dynamic Cost-Threshold

By knowing that every starting query has an initial cost, we can suggest a maximum cost that must not be exceeded. For example, if the query costs are $\mathsf{cost}(\tau)$, all memorized variants should be cheaper than e. g. $\mathsf{cost}(\tau) \cdot \log(\mathsf{cost}(\tau))$.

The idea behind this heuristic lies in the understanding that one sorting or index building can help improving the query plan. For instance, by sorting a relation by an attribute, which can be used later on to take a cheaper operator, one can reduce costs, not only locally, but throughout the whole branch. Similar to Simulated Annealing, we have a tolerance that gives us the possibility to overcome local minima.

$$\tau_1 \underset{}{\overset{1}{\rightleftharpoons}} \tau_2$$

1) if $\mathsf{cost}(\tau_2) \leq \mathsf{cost}(\tau_1) \cdot \mathsf{d}$

$\mathsf{d} = 1$ delivers Hill Climbing
$\mathsf{d} > 1$ (e. g. 1.1 or $\log(\mathsf{cost}(\tau_1))$) delivers Simulated Annealing

5.3 Static Variation-Threshold

Beside cost control, the system must be able to limit the number of generated variants. There are two different possibilities for static limitation: 1) we memorize maximally x best variants, so that we have to drop in every step some variants, in order to be able to look for new ones (implicitly after each rule application an evaluation $[...]_\mathsf{l}$ is computed), or 2) in every step we allow the best x variants to be memorized. The first idea gives the possibility to limit the search with a static threshold, so that - like in a breadth search - one just considers x variants by a kind of Hill Climbing approach. In the second case, there is a fixed number of x-th new variants so that one can control exponential growth to a linear one.

5.4 Dynamic Variation-Threshold

A dynamic variants-threshold limits the creation of variants in every step. As a consequence, the system has the possibility to produce more variants at a point where rules reduce the cost in a better way than using rules which do not improve the query plan at all. Thus the system gets the facility to enlarge the search space at a point where the chance is high to find a cheaper query plan.

6 Conclusion and Future Work

Having the related knowledge about the work presented in section 2 of this paper, it was our goal to overcome the difficulties and disadvantages of the, by now, developed systems. As a result, rule ordering is now quite flexible and controllable. The combination of cost-based, statistic-based and heuristic-based dependencies of ordering, gives us much more power to find the optimal execution plan.

Thus we are able to experiment with different strategies of rule ordering and global control to validate and improve heuristics.

The experiments, so far, show that the heuristical pruning methods should only be considered when having a larger numbers of joins (≥ 12 joins). Since then, it has been sufficient to narrow the search only by given conditions, so that no heuristics at all are necessary during the optimization procedure. The reason for this is that the search space is adequately small enough, e. g. 40 variants by 5 relations and 2 selections. A more important point is the ordering procedure of rules. Grouping commutative and associative joining rules with their physical implementations cuts the amount of needed applicable rules to a minimum. Parallel execution of these rules with selection resp. projection ordering, semijoin rewritings, and generation sorting and indexing rules, further rarefies the provisional results.

The upcoming work will concentrate on testing our approach, not only comparing it with common term rewriting systems, but with main database optimizers, like Oracle, DB2 or SQLServer.

Acknowledgments. I would like to thank my supervisor Udo W. Lipeck for his helpful discussions and propositions.

References

[1] L. Becker and R. H. Güting. Rule-based optimization and query processing in an extensible geometric database system. *ACM Trans. Database Syst.*, 17(2):247–303, 1992.

[2] M. Cherniack and S. B. Zdonik. Changing the rules: Transformations for rule-based optimizers. In *SIGMOD Conf.*, pages 61–72, 1998.

[3] G. Graefe and D. J. DeWitt. The exodus optimizer generator. In *SIGMOD Conf.*, pages 160–172, 1987.

[4] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *ICDE*, pages 209–218, 1993.

[5] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. In *SIGMOD Conf.*, pages 377–388, 1989.

[6] A. Heuer and J. Kröger. Query optimization in the croque project. In *DEXA '96: Proc. of the 7th Int. Conf. on Database and Exp. Systems Applications*, pages 489–499, London, UK, 1996. Springer-Verlag.

[7] N. Kabra. *Query optimization for object-relational database systems.* PhD thesis, 1999. Supervisor-David J. Dewitt.

[8] N. Kabra and D. J. DeWitt. Opt++: An object-oriented impl. for extensible database query optimization. *VLDB J.*, 8(1):55–78, 1999.

[9] A. Kemper and G. Moerkotte. Adv. query processing in object bases using access support relations. In *VLDB '90: Proc. of the 16th Int. Conf. on Very Large Data Bases*, pages 290–301, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[10] J. Kröger, R. Illner, S. Rost, and A. Heuer. Query rewriting and search in croque. In *ADBIS '99: Proc. of the 3rd East Europ. Conf. on Adv. in Databases and Inf. Systems*, pages 288–302, London, UK, 1999. Springer-Verlag.

[11] J. Kröger, S. Paul, and A. Heuer. On the ordering of rewrite rules (extended abstract). In *ADBIS '98: Proc. of the 2nd East Euro. Symp. on Adv. in Databases and Inf. Systems*, pages 157–159, London, UK, 1998. Springer-Verlag.

[12] U. W. Lipeck and D. Mantel. Matching cartographic objects in spatial databases. In *ISPRS '04*, pages 172–176, 2004.

[13] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in starburst. In *SIGMOD Conf.*, pages 39–48, 1992.