New Algorithms for Lexical Query Optimization

Nikita Mendkovich
Institute for System Programming of the Russian Academy of Sciences, 109004, Moscow, Alexandra Soljenitsina, 25,
mend@rambler.ru,

Sergey Kuznetsov
Institute for System Programming of the Russian Academy of Sciences, 109004, Moscow, Alexandra Soljenitsina, 25,
kuzloc@ispras.ru

Abstract. *New algorithms for query modifications are proposed. These algorithms involve lexical optimization based on mathematical transformations that have never been used for query optimization before.*

Keywords. Query optimization, lexical optimization.

1. Introduction

The standard problem of DBMSs usage is a lack of efficiency and low rate of handling of content-based access to data. The acceptable level of system performance may be achieved by query optimization which is a special part of database research. To make further discussion more clear it would be useful to define some terms used below.

By *query* we mean any kind of language expression that describes a multitude of data which is to be extracted from current database. In any query we mark *restriction* – a part of query that contains a multitude of terms describing data to be retrieved from a database. *Condition* is a part of restriction that is built upon a multitude of attributes of, countable and uncountable constants and contains not more than one comparison operation. *The goal of query optimization* is to minimize the system response time or/and the resource usage for a given output.

Usually there are 4 main steps of query optimization process marked up [5]:

1)query transformation to standard representation form;

2)modification, i.e. query transformation to a form which is the most efficient for further query evaluation;

3)choice of low-level procedures to be used during query evaluation;

4)optimal access plan generation.

In this paper we are going to investigate the modification part of optimization process, so we will briefly describe the main approaches that have been developed in this area. We will outline three main subparts of the modification step. They are - standardization, lexical and semantic optimizations.

Theoretical framework of lexical optimization subpart has been described for the first time in P. Hall's paper [4], where the optimization procedure based on the rules of relational algebra has been suggested. In general lexical optimization may be defined as query redundancy elimination *via* joining up existing conditions or removing redundant ones by detection of common sub-expressions, transitive rules use, etc. In contrast to other optimization procedures, lexical optimization involves only analysis of query restriction as a lexical construction and a comparison between different conditions that they contain. An example of lexical algorithms is well known magic sets transformation method [1, 6], oriented to modify the original program by means of additional "magic" conditions, which narrow the computation to what is relevant for answering the query. For more detailed observation of papers on query optimization see [2, 5].

The goal of optimization transformations represented in this paper is elimination of the redundant conditions, which may result in faster query evaluation by a DBMS. Such algorithms are used in Oracle query optimizer [8] but they are trivial and may be useful in rare cases only. In this paper we present the optimization transformations based on mathematical transformation algorithms that have never been used before for query optimization procedure.

2. Lexical algorithms for query optimization

2.1. Algorithm of query standardization

The main problem of lexical optimization algorithms of any kind, beginning from those suggested by Hall [4], is a providing an automatic comparison of different conditions in query body. Such automatic comparison requires a standardization of the conditions to detect semantically equal conditions independently of the way they are represented. Since the nested query problem seems to be solved (see [3, 7]), below we will discuss only the problem of standardization of simple condition expressions such as:

arith-expr1 comp-op arith-expr2,

where *arith-expr1* is an arithmetical expression containing database column names, *arith-expr2* – a constant expression, *comp-op* – a comparison operation.

Standardization algorithm's steps are:

1)all brackets in a given expression should be opened because otherwise the final representation might be irregular or would demand complication of the algorithm. Final expression should be a sum of single elements, constant values or attributes, multiplications, where attributes work as variables.

2)all sub-expressions containing character symbols should be moved to the left part of the comparison (*arith-expr1*), all constants – to the right part (*arith-expr2*). Comparison operations should be standardized as «more» (>), «equal» (=) and «less» comparison (<). Negations should be represented by opposite expression.

3)elements of multiplications and multiplications themselves should be transposed in any order (the most obvious one seems to be based on ASCI-coding).

4)all possible mathematic operations used in expressions should be evaluated. For example a*x*x sub-expression goes over to a*x^2 and *arith-expr*2 will become one constant value.

After optimization the final expression may be transformed back to the form containing brackets.

Next step means optimization of query restriction defined by a logical expression. The expression should be transformed into equivalent expression or a sequence of expressions that are shorter and easy to evaluate. We should eliminate not only completely equal conditions but also those with equal *arith-expr1* parts. For example, if we find out two conditions in two different expressions such as A>B and A>C where B>C, the A>B should be "absorbed".

For instance, let's consider the following SQL-query.

Example 1:

```
SELECT User_ID
FROM Users
WHERE (ID>0 AND Salary>500 AND Salary+Bonus<1000) OR (Salary>400)
```

Obviously, restriction of this query can be represented as "Salary>400" because first conjunction contains "Salary>500". And, if the value of the "Salary" attribute should be more then 500, it would also be more then 400. Therefore, the whole multitude of data, which corresponds to the first conjunction of conditions, will be contained in multitude described by "Salary>400".

To realize an "absorption" transformation as algorithm all pairs of conjunctions that contain one or more conditions with the same *arith-expr1* and *comp-op* ("absorptionable") should be identified. Other conditions should be equal in both conjunctions or one of them should contain only absorptionable conditions as in example 1.

Rules of the "absorption" transformation are described in the Table 1 below. First row contains possible views of absorbing condition, first column – views of condition which should be absorbed. Cells of the table 1 contain relation of *arith-expr2* necessary to execute "absorption". If cell contains "-", it means that "absorption" is impossible in this case.

Table 1. Rules of absorption.

	y=const2	y>const2	y<const2
x=const1	const2= const1	const1> const2	const2> const1
x>const1	-	const1 > const2	-
x< const1	-	-	const2< const1

After "absorption" the conjunction which contains absorbed condition should be eliminated. In example 1 it is "ID>0 AND Salary>500 AND Salary+Bonus<1000"

This easy reduction procedure enables us to eliminate 3 conditions out of 4.

2.2. Quine optimization algorithm

In case one deals with more complicated task: a group of 2 or more conjunctions contents more then two conditions with equal *arith-expr1* part, "Quine algorithm"[9] should be used to search prime implicants of truth functions. For example, a query selecting data on authors based on information about publications and citations (Example 2).

Example 2:

WHERE Articles>25 AND Books>2 AND Citations=32
OR NOT(Articles>25) AND Books>2 AND Citations=32
OR NOT(Articles>25) AND Books>2 AND NOT(Citations=32)
OR Articles>25 AND NOT(Books>2) AND NOT(Citations=32)

Without negations it would be represented as a set of conjunction:

1)Articles>25 AND Books>2 AND Citations=32
2)OR Articles=25 AND Books>2 AND Citations=32
3)OR Articles<25 AND Books>2 AND Citations=32
4)OR Articles<25 AND Books>2 AND Citations>32
5)OR Articles<25 AND Books>2 AND Citations<32
6)OR Articles=25 AND Books>2 AND Citations>32
7)OR Articles=25 AND Books>2 AND Citations<32
8)OR Articles>25 AND Books=2 AND Citations<32
9)OR Articles>25 AND Books=2 AND Citations>32
10)OR Articles>25 AND Books<2 AND Citations<32
11)OR Articles>25 AND Books<2 AND Citations>32

Pairs of constructions such as (A>B)&X and NOT(A>B)&X, where X is the common conjunct are to be marked up. The list of detected X implicants should be saved and set at the first column (see table 1). Numbers of all conjuncts are represented at the first row. «1» in the table's cells means that the implicant is included in expression represented in that column.

After that optimizer makes the following actions.

1)excludes all columns that content only zero values (in this case 8, 9, 10, 11);

2)marks all columns with only one non-zero value, corresponding rows are *substantial implicants*. If not all the implicants are substantial, optimizer chooses any (in this case shortest) multitude that corresponds to all columns.

3)identifies all columns which contain the same set of values. All columns-"duplicates" except one are to be eliminated.

4)eliminates all rows which do not content non-zero values.

5)selects set of implicants that contain non-zero values in all columns that still exist. If there are few possible sets optimizer chooses the one which contains a minimal number of conditions.

Table 2. Representation of restriction from example 2.

	1	2	3	4	5	6	7	8	9	10	11
Books>2 AND Citations =32	1	1	1	0	0	0	0	0	0	0	0
Articles= 25 AND Books>2	0	1	0	0	0	1	1	0	0	0	0
Articles< 25 AND Books>2	0	0	1	1	1	0	0	0	0	0	0

Final view of restriction will include all conjuncts selected at step 1, and implicants selected at steps 2 and 5. After Quine optimization for Example 2 the final view (with usage of negations) will be:

Books>2 AND Citations=32
OR NOT(Articles>25) AND Books>2
OR Articles>25 AND NOT(Books>2) AND NOT(Citations=32)

That means 50% of conjunctions and 58,3% of corpuscular conditions have been eliminated.

2.3. Optimization algorithm for set of expressions

The most of papers dedicated to query conditions optimization do not discuss attributes sets contained in the conditions. We suggest to represent elemental conditions as linear expressions or inequalities like:

$A_{i1}X_1+\ldots+A_{in}X_n$ *comp-op arith-expr2*,

where $X_1 \ldots X_n$ are mathematical expressions containing only unknown variables (attributes) and A_{ij} (where *i* is a number of equation and *j* – a number of X) – *expression index*, constant value.

Analysis of the restriction as a system of expressions makes it possible to simplify it by replacement of the set of the expressions containing the identical information by the only expression. The reduction procedure is based on linear algebra tools. The Example 3 illustrates the procedure.

Example 3

WHERE Salary+Bonus>10000
AND Salary-Payment>3000
AND Payment-Bonus<100
AND 2*Payment>8000

First we are to describe system by the table (Table 2). Any attribute of the expression contained in the restriction acts as a variable and contributes to the corresponding table's column. Any row corresponds to expression (condition). Any table's cell contains a constant value that serves as an index of variable. For the condition "Salary-Payment>3000" the index for "Salary" is "1", for "Payment" – "-1", for "Bonus" which is not mentioned in this condition the index is "0". Last row contains *arith-expr2* values.

Table 3. Representation of expression system from example 3.

Salary	Bonus	Payment	*Comp-op*	*arith-expr2*
1	1	0	>	10000
1	0	-1	>	3000
0	-1	1	<	100
0	0	2	>	8000

Algorithm described below executes two expression transformations that do not change system sense:

-expression multiplication by arbitrary index;

-development of a new expression (conclusion) on ground of two expressions.

One row which contains the largest number of non-zero variables (let it be *N*) is to be selected. We call it *working row* or *working expression*. On the next step we construct a sub-system of expressions which contains working expressions and all rows containing non-zero values only at the same cells that working row does.

If a number of variables in working expression is N, then the number of selected expressions should be N as well. If the number of selected expressions is less then N, we treat a group of variables working expression variables as with single variable. For example, for sub-system:

a+b+c>const1
a<const2;

where "a", "b" and c are attributes, then we treat "b+c" as the single variable because non-working expression describes only "a" variable.

For working expression and subsystem expression a set of *Y relation* is calculated. For expressions *i* and *j* for all columns we can calculate Y from equation A_{il}-A_{jl}*Y=0, where *l* is a column number. If both rows have non-zero values at the same cells and values of Y are equal for all columns of two rows being compared, then one of them should be eliminated as a semantic duplicate.

All pairs of working and non-working expression can have two standard forms.

Form 1:

a±b *comp-op* const1
a *comp-op* const2.

The first expression in Form 1 is a working one, "const1" and "const2" are constant values, "±" is a sign of addition or subtraction. "a" is an attribute or a sum of attributes common for both of expression, "b" is a attribute or a sum of attributes which are contained only in working expression.

Table 4. Rules for conclusions development for expressions represented in view 1.

Working expression	***a+b>const1***	***a+b<const1***
b>const2	-	a< const1-const2
b<const2	a> const1-const2	-
a>const2	-	b< const1-const2
a<const2	b> const1-const2	-
Working expression	***a-b>const1***	***a-b<const1***
b>const2	a> const1+const2	-
b<const2	-	a< const1+const2
a>const2	b> const2-const1	-
a<const2	b< const2-const1	-

To develop a conclusion on the base of such expressions' pair the rules represented in table 4 are to be used. In the cells of this table possible conclusions developed from the pair of working and non-working expressions by the procedure described in Table 4 are represented in the cells of the table. Symbols "a" and "b" are attributes or a sum of values, "-" means that conclusion for this two expressions cannot be developed.

Table 5. Rules for conclusions development for expressions represented in view 2.

Working expression	***(a+b>const1)***	***(a+b<const1)***
a+c>const2	-	c-b> const2-const1
a+c<const2	c-b< const2-const1	-
a-c>const2	-	b+c< const1-const2
a-c<const2	b-c> const1-const2	-
c-a<const2	-	b+c< const2+const1
c-a>const2	b+c> const2+const1	-
c-b>const2	a+c> const2+const1	-
c-b<const2	-	a+c< const2+const1
b-c>const2	-	a+c< const1-const2
b-c<const2	a+c>const1-const2	-
Working expression	***a-b>const1***	***a-b<const1***
a+c>const2	-	b+c> const2-const1
a+c<const2	b+c< const2-const1	-
a-c>const2	-	b-c> const2-const1
a-c<const2	b-c< const2-const1	-
c-a<const2	-	c-b< const2-const1
c-a>const2	c-b> const2+const1	-
c-b>const2	-	c-a> const2-const1
c-b<const2	c-a< const2-const1	-
b-c>const2	a-c> const2+const1	-
b-c<const2	-	a-c< const2+const1

Form 2:
a±b *comp-op* const1
a±c *comp-op* const2,

The first expression in Form 2 is a working one, “const1” and “const2” are constant values, “±” is a sign of addition or subtraction. Symbol “a” is an attribute or a sum of attributes common for both expressions, “b” and “c” are attributes or sums of attributes which are contained only in one expression.

To develop a conclusion on the base of such expressions’ pair the rules represented in table 5 are to be used.

Subsystems of working expressions are developed one after another until all expressions will be used as working or non-working expressions.

At the next stage every expression is compared with all conclusions that (i) do not belong to its own subsystem and (ii) whose *arith-expr1* contains set of attributes of this expression.

If expression and conclusion conjunction is false, the query execution may be finished because the restriction is impossible. If selected conclusions describe all attributes of the expression and their conjunction is true, the expression can be eliminated as semantic duplicate.

No conclusions are to be included into final form of a restriction.

For Example 3 pairs of conditions “2*Payment>8000” and “Salary-Payment>3000” develop conclusion “Salary>7000”. “Salary+Bonus>10000” and “Salary>7000” means “Bonus<=3000”. “Payment-Bonus<100”, “2*Payment>8000” and “Bonus<=3000” conjunctions are false. So query in Example 3 execution is to be finished.

It should be noted, that in some cases beginning query form can be optimal and may need no lexical optimization. At the same time, query expression standardization is unavoidable for any query modification process and avoidance of it means rejection of this kind of optimization at all. In the process of suggested algorithms optimization to the check-up of the optimization necessity is executed at the early stage to avoid redundant operations.

3. Conclusions

Three algorithms are proposed for SQL query optimization. They provide effective simplification of restrictions containing complex multi-attribute conditions by exclusion of redundant and false conditions. The developed algorithms have low “optimization risks”.

References

[1] Faber W, Greco G, Leone N. Magic Sets and their application to data integration. Journal of Computer and System Sciences 2007; 73(4): 584-609.

[2] Chaudhari S. An Overview of Query Optimization in Relational Systems. Microsoft; 1998. http://research.microsoft.com/pubs/76059/pods98-tutorial.pdf [12/01/2007]

[3] Ganski R, Wong H. Optimization of Nested SQL Queries Revisited. In: Umeshwar D., Traiger I. L. editors. Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data Annual Conference 1987; San Francisco, California, USA; May 27-29, 1987. SIGMOD, 16(3), 1987. 22-33.

[4] Hall PAV. Optimization of single expressions in a relational data base system. IBM Journal Res. Devel. 1976. 20(3). 244-257.

[5] Jarke M, Koch J. Query Optimization in Database Systems. ACM Computer Survey 1984, 16(2). 111-152.

[6] Mumick IS, Finkelsteint SJ, Pirahesh H, Ramakrishnan R. Magic is Relevamt. In Garcia-Molina H. Jagadish, editors. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data; May 23-25, 1990; Atlantic City, NJ, USA. ACM, 1990. p. 247-258.

[7] Muralikrishna M. Improved Unnesting Algorithms for Join Aggregate SQL Queries Having Aggregates. In Li-Yan Yuan, editor. Proceeding of the 18th International Conference on Very Large Data Bases; 1992, August 23 – 27; Vancouver, Canada. Morgan Kaufmann, 1992. p. 91-102.

[8] Query Optimization in Oracle Database 10g Release 2. An Oracle White Paper, June 2005. Oracle; 2005. http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_general_query_optimization_10gr2_0605.pdf [06/30/2008]

[9] Quine WV. On cores and prime implicants of truth functions. American Mathematics Monthly 1959; 66(9). 755-760.