Rough Optimizations of Complex Expressions in Infobright's RDBMS

Dominik Ślęzak[1,2], Piotr Synak[2], Jakub Wróblewski[2],
Janusz Borkowski[2], and Graham Toppin[3]

[1] Institute of Mathematics, University of Warsaw,
ul. Banacha 2, 02-097 Warsaw, Poland
[2] Infobright Inc., Poland,
ul. Krzywickiego 34, lok. 219, 02-078 Warsaw, Poland
[3] Infobright Inc., Canada,
47 Colborne St., Suite 403, Toronto, ON M5E1P8 Canada
{slezak,synak,jakubw,januszb,toppin}@infobright.com

Abstract. We discuss the importance of analytic SQL statements with complex expressions in the business intelligence and knowledge discovery applications. We report the recent improvements of the execution of complex expressions in the Infobright's RDBMS, which is based on the paradigms of columnar databases and adaptive rough computations over the granulated metadata layer.

Keywords: RDBMS, Analytic SQL, Complex Expressions, Rough Computing.

1 Introduction

An important trend in the IT industry relates to the RDBMS solutions specialized in database analytics, aimed at advanced reporting and ad-hoc querying against massive amounts of data. Infobright's technology[1] is an example of such a solution, optimized particularly with regard to the analysis and exploration of rapidly growing machine-generated data sets[2]. Infobright's approach to solving the underlying computational scalability problems is based on a specific application of rough computing [1] in combination with the principles of columnar databases [2]. An important ingredient here is a layer of fast heuristic algorithms that attempt to minimize and optimize the data access during the query execution [3]. There is an ongoing process of improving this layer by means of new ideas and implementations. This paper reports one of the areas of such improvements, dedicated to SQL statements with complex expressions.

We concentrate on complex expressions that can be interpreted as dynamically derived columns of the original or intermediately created tables involved into the analytic SQL statement execution. Such expressions may occur in the conditions (e.g.: `WHERE expression = ...`), aggregations (e.g.: `SELECT SUM(expression)`), groupings (e.g.: `SELECT expression as A ... GROUP BY A`) et cetera. They may take a form of date functions, arithmetic operations, conditional expressions et cetera

[1] `www.infobright.org, www.infobright.com`
[2] `en.wikipedia.org/wiki/machine-generated_data`

J.T. Yao et al. (Eds.): RSCTC 2012, LNAI 7413, pp. 94–99, 2012.

[4]. They may be also used to combine the results of the aggregate functions and they may involve or be involved in the correlated subqueries [5].

The queries with expressions often occur in business intelligence and decision making applications [6]. The string functions may be useful in the analysis of the above-mentioned machine-generated data, where web logs, url addresses et cetera take a form of long varchar columns [7]. Expressions are also important in the knowledge discovery applications. Consider, e.g., the task of the feature extraction [8], where the SQL-based scripts can derive new useful attributes from a relational database [9]. As another example, consider the methods of SQL-based machine learning, such as the decision tree construction proposed in [10], which can be further extended by searching for the cuts on linear combinations of the original columns [11]. In all such cases, expressions represent the new dimensions of a decision model. The resulting model can be represented by expressions too, e.g.: boolean expressions defining the root-to-leaf paths in a tree, or arithmetic expressions defining the clusters of homogeneous rows.

In this paper, we show how to optimize the Infobright's process of resolving the analytic SQL statements with expressions. In Section 2, we recall the foundations of the discussed database architecture, emphasizing its relationships to rough approximations. In Section 3, we outline the drawbacks of our previous implementation of expressions. We follow up with an overview of possible improvements and the solution introduced in the latest version of our product. Section 4 concludes the paper.

2 Infobright's Architecture

Infobright's RDBMS is an example of a rough-columnar database engine. It combines the columnar data storage/processing [2] with the metadata layer containing the granulated tables used to minimize the data access intensities, basing on the principles of rough sets [1]. Rows of a granulated table (further called rough rows) correspond to some partition blocks of the original data rows. Columns of a granulated table (further called rough attributes) store statistics (further called rough values) describing the original data columns within particular blocks of rows. The rough values may contain information such as min/max values (interpreted specifically for different data types), a sum of values (or, e.g., a total length of strings), a number of nulls et cetera.

New rough values are computed after receiving each block of 2^{16} freshly loaded rows. The gathered rows correspond to a new rough row in the granulated table. The engine first decomposes rows onto particular columns' values and creates data packs – collections of 2^{16} values of each column. In other words, each data pack corresponds to a single block of rows and a single data column. Each data pack is represented by a rough value summarizing its statistics at a metadata level. Last but not least, each of data packs can be compressed independently. Figure 1 illustrates the overall framework of loading, storing and querying the data. As displayed, rough values can be applied to categorize some data packs as not requiring access with respect to the query conditions. However, rough values can assist also in resolving other parts of SQL clauses, such as aggregations, different forms of joins, correlated subqueries et cetera [3].

Sticking with the example of the `WHERE` clause, there are two cases when the given portion of data can be categorized as not requiring further access. The first case is to

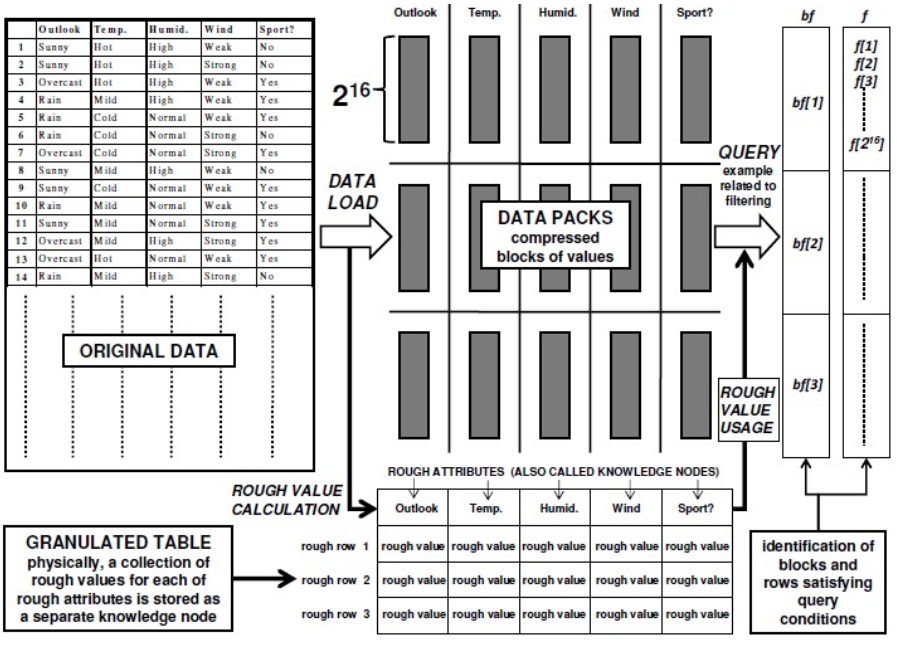

Fig. 1. Loading, storing and querying in Infobright's RDBMS. The metadata layer containing the granulated tables is often called the Infobright's knowledge grid. The contents of rough attributes are stored in so called knowledge nodes. During the query execution, all applicable knowledge nodes are put into memory. Rough values may be used, e.g., to qualify (disqualify) the blocks of rows that satisfy (do not satisfy) the given `WHERE` clause. Only the borderline blocks (precisely, their data packs corresponding to the columns used in the clause) need to be decompressed and examined row by row. The filtering results obtained at the level of blocks and single rows are passed as the arrays denoted by bf and f to the further stages of the query execution.

use rough values against `WHERE` clauses in order to eliminate the blocks that are for sure out of the scope of a given query [12]. The second case occurs when it is enough to use a given block's statistics. It may happen, e.g., when we are sure that all rows in a block satisfy query conditions and, therefore, some of its rough values can represent its contribution into the final result. In our research, we noted an analogy between the two above-discussed cases and the positive/negative regions defined within the theory of rough sets [1]. It helped us to employ various AI-based heuristics aimed at minimizing the amounts of data packs that need to be accessed, i.e., the data portions that cannot be categorized as positive/negative by using their corresponding rough values.

Infobright's performance depends on the quality of rough values and the algorithms that use them. It is worth adding that we assign rough values not only to physical data packs but also to intermediate structures generated during the query execution (e.g.: hash tables used in aggregations). Also, we dynamically produce rough values applicable at further execution stages. This will be better visible in the next section, where we explain why it is worth computing such new rough values for expressions.

3 New Implementation of Complex Expressions

The presented approach enables Infobright to gather positive feedback in the areas such as online analytics, financial services and telecommunications, with the largest customers approaching one petabyte of the processed data. Besides rough operations, the speed of analytic queries is assured by the already-mentioned advantages of columnar database design [2], as well as the adaptive query optimization [13] and parallel execution of data operations [14]. On the other hand, a challenge in front of technologies combining many architectural aspects is to assure stable performance, i.e., comparable execution speed of the SQL statements differing only in details.

One of our problems with regard to the above-mentioned performance stability used to refer to the way of processing complex expressions when compared to the original data columns. As already stated, expressions can occur in many places of the SQL syntax, including filters, attributes to be aggregated et cetera. On the other hand, for the users of the business intelligence and data mining tools, it is quite natural to expect that the speed of execution of the analytic queries does not decrease significantly when replacing a single column by an expression involving several columns. Using a simple example, the users realize that execution of the clause `SELECT X+Y as A ... GROUP BY A` takes longer than `SELECT X ... GROUP BY X`, but they will not accept an order of magnitude difference in the query execution time.

Prior to the recent 4.1 release of Infobright's RDBMS, the default way of processing expressions relied on the MySQL code. Basically, we treated expressions as additional data columns with dynamically derivable values and we internally used the MySQL structures to compute a given expression's value for each particular row. Interpretation of expressions as such additional columns may be actually compared to creation of features of information systems corresponding to the physical data tables or intermediate (although not necessarily materialized) tables resulting from joins, orderings, groupings et cetera, treated as dynamically derived information systems [15].

Such a strategy enabled us to quickly extend the query functionality onto all expressions implemented within MySQL. On the other hand, there were three issues: 1. Computation based on MySQL structures was slightly slower than expected; 2. The MySQL code turned out as not fully thread safe [16], which limited our abilities to parallelize some steps of the execution of SQL statements with expressions [17]; 3. There was no rough computation support for the dynamically derivable columns corresponding to expressions, i.e., using the above example of query `SELECT X+Y as A ... GROUP BY A`, the rough values for `A` did not exist, so they could not help in optimizing the access to data packs of the underlying data columns `X` and `Y`.

In order to solve the two first problems, we re-implemented the code for computing complex expressions by ourselves. Although time consuming, this development project was finished successfully, providing the functional coverage for a wide range of expressions useful in the database analytics. However, addressing the challenges only at the level of the row-by-row calculations was not sufficient to obtain the performance comparable to queries with no expressions. Therefore, some algorithms extracting rough values of complex expressions for particular blocks of rows or, more generally, portions of data involved in the query execution turned out to be necessary.

Table 1. Functions & operators optimized at the rough level in Infobright 4.1

Logical
=
<>, ! =
<=
<
>
>=
BETWEEN
IN
NOT, !
AND, &&
OR
XOR
NOT BETWEEN
COALESCE
NOT IN
ISNULL
CASE
IF
IFNULL
NULLIF
IS NULL
IS NOT NULL

Numerical
+
−
*
/
%
DIV
ABS
EXP
LOG10
LOG2
LOG
SQRT
FLOOR
BIN
OCT

String
LIKE
NOT LIKE
CONCAT
LENGTH
SUBSTR
SUBSTRING
INSTR
LOCATE
LEFT
MID
RIGHT
LOWER
LCASE
UPPER
UCASE

Others
INET_NTOA

Date/Time
CURDATE
CURRENT_DATE
CURRENT_TIME
DATE
DATEDIFF
DAY
HOUR
MONTH
YEAR
CURRENT_TIMESTAMP
CURTIME
DAYOFMONTH
DAYOFYEAR
LOCALTIME
LOCALTIMESTAMP
EXTRACT
MINUTE
NOW
QUARTER
SECOND
TIME
TO_DAYS

There are various potential ways of providing rough values of expressions to the Infobright's algorithms minimizing the data access. For instance, a popular research trend in the database industry is to use the statistics of the occurrence of expressions in the query logs. In our case, it would mean memorizing the knowledge nodes of some of dynamically derived columns corresponding to the occurred expressions and reusing them in the future queries. However, the question then arises how to maintain the right expressions, so the overall size of the metadata layer does not grow in an uncontrolled way. This is especially difficult to address in the case of ad-hoc query workloads, where each next statement may contain slightly different expressions.

We chose another solution, which is implementing at the rough level the selected group of popular types of expressions. Table 1 displays the functions and operators, which are supported in this way in the Infobright 4.1 release. This means that for a given block of rows or, in other words, its corresponding rough row in the granulated table the rough value of a new attribute corresponding to the complex expression is computed basing on the rough values of data columns involved in the expression's definition. Such dynamically created rough values are used by the query optimization and execution algorithms exactly in the same way as the rough values of the original data columns. The way of computing rough values for particular types of expressions turned out to be an interesting research task, closely related to the principles of granular and interval computing [18]. The tests conducted over real-world data sets prove that this strategy allows us to achieve the wanted stability of the query performance.

4 Conclusions

We presented the recently improved Infobright's implementation of SQL statements with expressions. It is based on computing the metadata statistics for attributes corresponding to expressions, so they can be processed similarly to the data columns.

References

1. Pawlak, Z., Skowron, A.: Rudiments of Rough Sets. Information Sciences 177(1), 3–27 (2007)
2. White, P., French, C.: Database System with Methodology for Storing a Database Table by Vertically Partitioning all Columns of the Table. US Patent 5,794,229 (1998)
3. Ślęzak, D., Eastwood, V.: Data Warehouse Technology by Infobright. In: Proc. of the 2009 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD), pp. 841–846 (2009)
4. Pöss, M., Nambiar, R.O., Walrath, D.: Why You Should Run TPC-DS: A Workload Analysis. In: Proc. of the 33rd Int. Conf. on Very Large Data Bases (VLDB), pp. 1138–1149 (2007)
5. Synak, P.: Rough Set Approach to Optimisation of Subquery Execution in Infobright Data Warehouse. In: Proc. of the Int. Workshop on Soft Computing for Knowledge Technology (SCKT), Hanoi University of Technology (2008)
6. Davenport, T.H., Harris, J.G.: Competing on Analytics - The New Science of Winning. Harvard Business School Press (2007)
7. Ślęzak, D., Toppin, G.: Injecting Domain Knowledge into a Granular Database Engine: A Position Paper. In: Proc. of the 19th ACM Conf. on Information and Knowledge Management (CIKM), pp. 1913–1916 (2010)
8. Liu, H., Motoda, H. (eds.): Feature Extraction, Construction and Selection: A Data Mining Perspective. Kluwer Academic Publishers (1998)
9. Wróblewski, J.: Analyzing Relational Databases Using Rough Set Based Methods. In: Proc. of the 8th Int. Conf. on Information Processing and Management of Uncertainty (IPMU), Part I, pp. 256–262 (2000)
10. Nguyen, H.S., Nguyen, S.H.: Fast Split Selection Method and Its Application in Decision Tree Construction from Large Databases. International Journal of Hybrid Intelligent Systems 2(2), 149–160 (2005)
11. Nguyen, H.S.: From Optimal Hyperplanes to Optimal Decision Trees. Fundamenta Informaticae 34(1-2), 145–174 (1998)
12. Metzger, J.K., Zane, B.M., Hinshaw, F.D.: Limiting Scans of Loosely Ordered and/or Grouped Relations Using Nearly Ordered Maps. US Patent 6,973,452 (2005)
13. Deshpande, A., Ives, Z.G., Raman, V.: Adaptive Query Processing. Foundations and Trends in Databases 1(1), 1–140 (2007)
14. Hellerstein, J.M., Stonebraker, M., Hamilton, J.: Architecture of a Database System. Foundations and Trends in Databases 1(2), 141–259 (2007)
15. Pawlak, Z.: Information Systems - Theoretical Foundations. Information Systems 6, 205–218 (1981)
16. Oracle: Multithreaded Programming Guide (Beta). Online Documentation (2010)
17. Borkowski, J.: Performance Debugging of Parallel Compression on Multicore Machines. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 82–91. Springer, Heidelberg (2010)
18. Kreinovich, V.: Towards Faster Estimation of Statistics and ODEs Under Interval, P-Box, and Fuzzy Uncertainty: From Interval Computations to Rough Set-Related Computations. In: Kuznetsov, S.O., Ślęzak, D., Hepting, D.H., Mirkin, B.G. (eds.) RSFDGrC 2011. LNCS (LNAI), vol. 6743, pp. 3–10. Springer, Heidelberg (2011)