A Rough-Columnar RDBMS Engine – A Case Study of Correlated Subqueries

Dominik Ślęzak
University of Warsaw &
Infobright Inc., Poland
slezak@infobright.com

Piotr Synak
Infobright Inc., Poland
synak@infobright.com

Janusz Borkowski
Infobright Inc., Poland
januszb@infobright.com

Jakub Wróblewski
Infobright Inc., Poland
jakubw@infobright.com

Graham Toppin
Infobright Inc., Canada
toppin@infobright.com

Abstract

Columnar databases provide a number of benefits with regard to both data storage (e.g.: data compression) and data processing (e.g.: optimized data access, parallelized decompression, lazy materialization of intermediate results). Their characteristics are particularly advantageous for exploratory sessions and ad hoc analytics. The principles of columnar stores can be also combined with a pipelined and iterative processing, leading toward modern analytic engines able to handle large, rapidly growing data sets. In this paper, we show how to further enrich such a framework by employing metadata layers aimed at minimizing the need of data access. In particular, we discuss the current implementation and the future roadmap for correlated subqueries in Infobright's RDBMS, where all above-mentioned architectural features interact with each other in order to improve the query execution.

1 Introduction

One of the main challenges of today's enterprises is to integrate ad hoc reporting and analysis features within applications and services that they deliver. It becomes necessary to support the data mining processes and analytically intensive query workloads, including exploratory SQL statements, which were not anticipated during the database design stages. Moreover, users need to operate on rapidly growing data without delays caused by time consuming model re-optimizations related to evolving data or query patterns.

The discussed RDBMS technology relies on the principles of columnar databases [15], with an additional usage of an automatically generated metadata layer aimed at improving a flow of data accesses while resolving queries [12]. The columnar approach enables to apply (de)compression algorithms that are better adjusted to analytical query execution characteristics [16] and can significantly improve the process of assembling results of ad hoc queries [1]. On the other hand, an appropriate choice and usage of metadata may lead toward a better query plan optimization [7], which can also adaptively influence the process of a query execution [4].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In our approach, metadata is applied throughout the whole process of the query execution in order to minimize the intensity of data accesses. The content of each column is split onto collections of values of some consecutive rows. Each of the data packs created this way is represented by its basic statistics at the metadata level. This approach led us toward developing algorithms identifying data packs that are sufficient to complete particular query execution stages. The size of metadata grows linearly with the size of data and remains several orders of magnitude smaller. Metadata units corresponding to newly formed data packs are computed independently from (meta)data already stored in a database. Moreover, the whole framework makes it possible to combine the classical benefits of columnar engines with a more iterative data processing [3].

The paper is organized as follows: In Section 2, we explain how metadata describing blocks of rows can assist in speeding up a query execution. We show how to operate with approximations of query results and we refer to correlated subqueries as an illustration. In Section 3, we recall advantages of the columnar databases with regard to analytic queries. We also discuss the compression of data decomposed with respect to both columns and blocks of rows. In Section 4, we introduce the rough-columnar architecture of Infobright's RDBMS[1]. We show how the principles of the columnar databases and rough operations can complement each other. In Section 5, we discuss the current implementation and the roadmap for further improvements of correlated subqueries. We also provide more details about their usefulness in practice. Section 6 concludes the paper.

2 Rough Operations on Granulated Data

In this section we discuss an idea of using granulated tables – specific metadata descriptions of data tables in a relational model. Rows of a granulated table correspond to some partition blocks of the original data rows. Columns of a granulated table (further called rough columns) 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 string values), a number of nulls et cetera. They can be employed, e.g., as a support of a cardinality estimation during the query plan optimization, or as a metadata layer assisting in the query execution.

Although such framework can be compared to a number of approaches to data block-level indexing, the first realistic implementation of a granulated metadata support of a query execution seems to refer to Netezza's nearly ordered maps, also known as the zone maps [9]. Netezza's invention was to partition data into blocks of consecutively loaded rows, annotate each of such blocks with its min/max values with respect to particular data columns, and use such rough values against `WHERE` clauses in order to eliminate the blocks that were for sure out of the scope of a given query. In this paper, we focus on the technology developed by Infobright, which goes further by means of applied types of statistics, identification of cases where blocks of data do not need to be accessed, and data processing operations for which such identification can be taken into account.

There is also another, not so broadly studied case when a block of rows does not need to be accessed. It occurs when it is enough to use its 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 of blocks that do not require accessing and positive/negative regions used in the theory of rough sets to learn classification models [10]. It helped us to understand how to employ various AI-based heuristics for our own purposes. It also explains why we use terms such as a rough column and a rough value when introducing foundations of our approach. Throughout the rest of the paper, we will refer to the both of above cases as to the solved blocks of rows.

The efficiency of rough computations depends on an ability to classify blocks as solved by rough values. Quality of granulated tables understood in this way relies on regularities within the collections of column values in particular blocks. In the case of solutions such as Netezza or Infobright, blocks are assembled along the flow of rows loaded into a database. Thus, the highest quality is expected for rough values corresponding to columns

[1] www.infobright.com

that are loosely ordered. Even if one knows that some other columns are going to be more frequently queried, it is hard for users to accept any time consuming reorganizations of rows that would improve the quality of the corresponding parts of metadata. Fortunately, it is also possible to consider more on-the-fly strategies for re-ordering the loaded rows [11], basing on ideas borrowed from the data stream cluster analysis [2].

The rough values can be also computed dynamically, as descriptions of intermediate results of the query execution. As a case study, consider a correlated subquery: a query nested in some other query – called an outer query – which often operates on another table – called an outer table – and which is parameterized by some outer table's values. Correlated subqueries can occur in a number of SQL clauses. Here, for illustrative purposes, consider the following example of the `WHERE` clause of the outer query: `T.a<(SELECT MIN(U.x) FROM U WHERE U.y=T.b)`. If `U` is large, then the execution of the outer query on `T` may be time consuming. However, as pointed out in [14], one can quickly derive approximate answers to particular subqueries and, for each row in `T`, check whether the above condition could be successfully resolved by using such dynamically obtained rough values. Let us also add that such a mechanism of fast production of the rough values estimating the query results is available as so called rough SQL in the 4.x^{++} versions of Infobright's RDBMS[2].

3 Toward Columnar Storage / Processing

Columnar databases seem to be particularly useful in database analytics. Complex reports can be realized by highly optimized column scans. Ad hoc filters can be processed directly against particular columns or projections [13]. Only those values that are associated with columns involved in a query need to be processed. Some of components of results can be kept as non-materialized for a longer time during the query execution [1]. Since each column stores a single data type, it is possible to apply better adjusted compression algorithms. It is also possible to execute queries against not fully decompressed data [16]. Thus, for analytic queries, columnar databases are able to manage available resources efficiently. Also, e.g. with regard to minimizing the disk I/O, the columnar approaches turn out to have the same optimization goals as those described in Section 2.

The basic idea underlying integration of columnar and rough methodologies is to split data both vertically and horizontally. However, such a solution is worth considering also because of other reasons. First of all, it allows to combine the above advantages of columnar databases with pipelined mechanisms of the data processing [3]. Also, data compression techniques are then able to adjust better to smaller, potentially more homogeneous contents [15]. In general, a behavior of compression algorithms may vary not only with respect to different columns but also with respect to patterns automatically detected inside different collections of values.

Let us also note that one of important challenges is to efficiently compress and process alphanumeric collections. Our motivation to look at this issue arises from an analysis of machine-generated data sets, which grow extremely fast and often require ad hoc querying. Such data sets include web logs, computer events, connection details et cetera. They often correspond to long varchar columns. Interestingly, machine-generated data values often follow some semantic constructs that are difficult to detect even for sophisticated compression algorithms [6]. Accordingly, we designed interfaces allowing domain experts to express their knowledge about internal structures of values stored in varchar columns. We also developed methods that apply the acquired knowledge to improve both the alphanumeric data compression and the corresponding rough values' quality [8].

4 Infobright as a Rough-Columnar Engine

Infobright's RDBMS engine combines techniques presented in Sections 2 and 3 in order to provide efficient executions of analytic SQL statements over large amounts of data. New rough values are computed after receiving each block of 2^{16} freshly loaded rows. The gathered rows correspond to a new entry in the granulated table. The

[2] www.dbms2.com/2011/06/14/infobright-4-0/

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.

There are actually two metadata layers. The first one stores information about a location and status of each data pack. It also contains lower level statistics assisting in pack's decompression or, if applicable, translating incoming requests in order to resolve them against only partially decompressed pack's content. This layer is often referred to as a data grid. The second layer, called a knowledge grid, maintains statistics corresponding to granulated tables. Besides elements already mentioned in Section 2, it contains some more advanced structures resembling those used in other approaches [4, 7]. A common feature of all stored structures is that their sizes are orders of magnitude smaller than the sizes of the corresponding data packs, and that they are all designed to be used in rough operations. Also, the granulated tables residing in the knowledge grid are internally structured in a columnar fashion, so particular rough columns can be efficiently employed.

Infobright's knowledge grid can be applied to minimize portions of data required to resolve queries along the lines discussed in Section 2. However, the data access reduction process is now conducted at a more fine-grained level of particular data packs rather than blocks of rows. Rough values are applied against query components in order to identify the data packs that are solved at particular data processing stages. The meaning of solved data packs is analogous to solved blocks. However, within a single block of rows, classifications of the data packs corresponding to different columns may vary. For instance, in case of an ad hoc multi-column condition, some data packs can be identified as solved while others may require a more thorough analysis. However, while combining such classifications, more data packs can change their status to solved. The same may happen after examining some of data packs value by value. Generally, a number of solved data packs can grow during a given query execution, basing on additional information acquired iteratively [12].

In summary, the discussed rough-columnar approach has already enabled Infobright to gather positive feedback from users in areas such as online analytics, financial services and telecommunications, with the largest current customer approaching one petabyte of processed data[3]. Still, there are more opportunities to use the above methods for a faster analytic querying against large data. For example, we use rough values not only for purposes of identifying solved data packs but also for heuristic ordering of packs to be decompressed, as well as for grouping packs into data areas corresponding to better-defined processing sub-tasks. We assign rough values not only to physical data packs but also to intermediate structures produced during the query execution (e.g.: hash tables constructed during aggregations and joins). We produce rough values that can be applied at further execution stages (e.g.: already-mentioned rough execution of correlated subqueries or the recently implemented computation of rough values for expressions). Finally, we can adapt a number of useful techniques developed within the main stream research on columnar databases, as visible also in the next section.

5 A Roadmap for Correlated Subqueries

There are various approaches to the optimization of correlated subqueries [5]. Their importance is visible in several areas of business intelligence, such as the trend or predictive analysis. Complex nested queries may occur at the reporting stages focused on identifying anomalies and risk indicators. They may be useful also in applications requiring storage of large sets of hierarchical objects in a relational format. In many cases, e.g. for large metadata repositories, a dynamic reconstruction of such objects may be also supported by massive self-join operations or some recursive SQL extensions, if available. However, there are cases where the usage of correlated subqueries seems to be particularly convenient, if they perform fast enough. For instance, for a purpose of analysis of documents stored in an EAV-like type of format, the correlated subqueries may be applied to extract information about documents that posses some specific semantic and/or statistical features.

[3] www.infobright.com/customer/jdsu

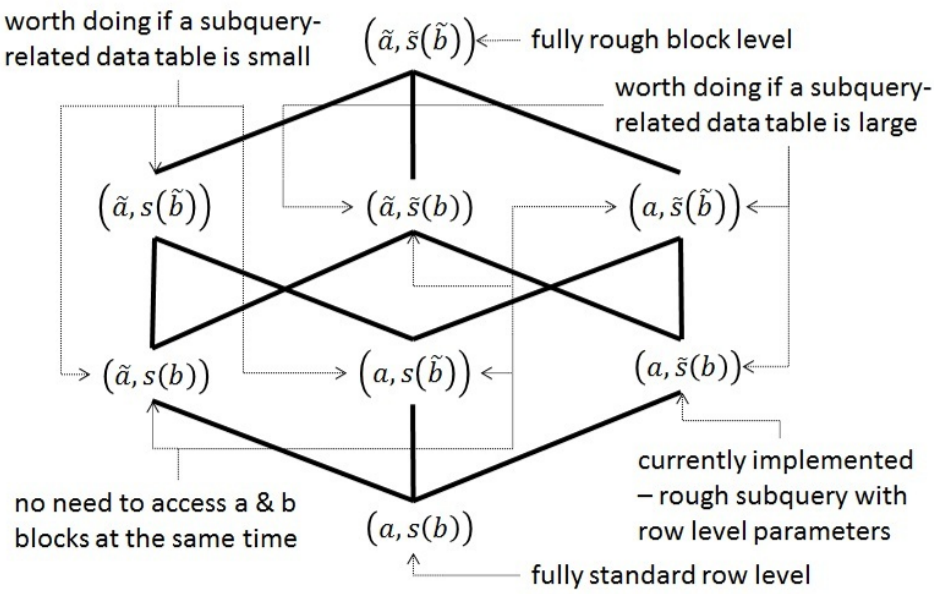

Figure 1: The roadmap for improving the execution of correlated subqueries. An example of a correlated subquery in the `WHERE` clause, where outer table's column a is compared with a result of subquery parameterized by outer table's column b. a and $\widetilde{a}$ denote, respectively, an exact value (for a row) or a rough value (for a block) of a. (The same applies to b and $\widetilde{b}$.) s and $\widetilde{s}$ denote, respectively, the standard and the rough execution of the subquery, which may be parameterized by b or $\widetilde{b}$. The pairs in brackets denote which modes of values and/or rough values are compared to each other while resolving the `WHERE` part of a query.

The current Infobright's implementation of the correlated subqueries was informally outlined in Section 2. Sticking to an example of a subquery in the `WHERE` clause, we launch its fast but inexact version with parameters induced by each consecutive row in the outer table and attempt to use its rough outcome to avoid the exact mode of execution. In Figure 1, it is illustrated by the cases $(a, \widetilde{s}(b))$ and $(a, s(b))$. The first of them symbolizes, for a given row, the comparison of its value on the column a with a rough result of the subquery parameterized by its value on the column b. If such a comparison is not sufficient to decide whether that row satisfies the `WHERE` clause, then, for that particular row, we need to proceed with the standard subquery execution $s(b)$.

Figure 1 represents also other cases waiting for implementation. Let us first consider $(\widetilde{a}, \widetilde{s}(\widetilde{b}))$, where we confront the rough value obtained as a result of $\widetilde{s}(\widetilde{b})$ with the rough value of a. If such a comparison fails to classify the block of rows as solved, the algorithm should proceed with either of: 1) the standard subquery execution $s(\widetilde{b})$ and a comparison of its result with the a's rough value; 2) decompression of the a's data pack and comparison of the a's values on particular rows with the previously computed $\widetilde{s}(\widetilde{b})$; 3) decompression of the b's data pack and comparison of the a's rough value with the rough result of $\widetilde{s}(b)$ computed for particular values of b. The possible computation strategies correspond to the top-down paths in Figure 1. For instance, the strategy $(\widetilde{a}, \widetilde{s}(\widetilde{b}))\rightarrow(\widetilde{a}, \widetilde{s}(b))\rightarrow(a, \widetilde{s}(b))\rightarrow(a, s(b))$ would proceed with the current implementation if the above-described step #3 fails to categorize some of the rows that are important for the final query result.

The choice of the computation strategy shall depend on the expected gain and cost related to the standard subquery execution and/or the decompression of data packs of particular columns. It can be decided for each of blocks of rows separately or for the entire table, depending on how the underlying column scans can be optimally arranged. Let us also note that in order to proceed with $s(\widetilde{b})$ and $\widetilde{s}(\widetilde{b})$, the subquery syntax needs to be modified. For instance, for a query considered in Section 2, its part `U.y = T.b` would need to be automatically replaced by `U.y BETWEEN T.b_min AND T.b_max`, where `T.b_min` and `T.b_max` denote the min/max statistics of b for a given block. This requires further research for a broader class of queries.

6 Conclusion

Our primary goal in this paper was to show that it is worth combining the ideas behind columnar databases and rough operations on granular metadata. We interpreted Infobright's solution as an example of a rough-columnar RDBMS. In particular, we investigated the execution of correlated subqueries in order to show a variety of possibilities provided by the coexistence of presented methods within the same framework.

References

[1] Abadi, D.J., Myers, D.S., DeWitt, D.J., Madden, S.: Materialization Strategies in a Column-oriented DBMS. In: Proc. of ICDE (2007) 466–475

[2] Aggarwal, C.C. (ed.): Data Streams: Models and Algorithms. Springer (2007)

[3] Boncz, P., Zukowski, M., Nes, N.: MonetDB/X100: Hyper-pipelining Query Execution. In: Proc. of CIDR (2005) 225–237

[4] Deshpande, A., Ives, Z.G., Raman, V.: Adaptive Query Processing. Foundations and Trends in Databases **1**(1) (2007) 1–140

[5] Elhemali, M., Galindo-Legaria, C.A., Grabs, T., Joshi, M.: Execution Strategies for SQL Subqueries. In: Proc. of SIGMOD (2007) 993–1004

[6] Inenaga, S., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S., Mauri, G., Pavesi, G.: On-line Construction of Compact Directed Acyclic Word Graphs. Discrete Applied Mathematics **146**(2) (2005) 156–179

[7] Ioannidis, Y.E.: The History of Histograms (Abridged). In: Proc. of VLDB (2003) 19–30

[8] Kowalski, M., Ślęzak, D., Toppin, G., Wojna, A.: Injecting Domain Knowledge into RDBMS – Compression of Alphanumeric Data Attributes. In: Proc. of ISMIS (2011) 386–395

[9] 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)

[10] Pawlak, Z., Skowron, A.: Rudiments of Rough Sets. Information Sciences **177**(1) (2007) 3–27

[11] Ślęzak, D., Kowalski, M., Eastwood, V., Wróblewski, J.: Method and System for Database Organization. US Patent Application 2009/0106210 A1 (2009)

[12] Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. PVLDB **1**(2) (2008) 1337–1345

[13] Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E.J., O'Neil, P.E., Rasin, A., Tran, N., Zdonik, S.B.: C-Store: A Column-oriented DBMS. In: Proc. of VLDB (2005) 553–564

[14] Synak, P.: Rough Set Approach to Optimisation of Subquery Execution in Infobright Data Warehouse. In: Proc. of SCKT (PRICAI Workshop) (2008)

[15] 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)

[16] Zukowski, M., Héman, S., Nes, N., Boncz, P.: Super-scalar RAM-CPU Cache Compression. In: Proc. of ICDE (2006) 59