Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1455–1462

Enhanced Rough SQL for Correlated Subqueries

Marcin Kowalski*†, Dominik Ślęzak*† and Piotr Synak†
*Institute of Mathematics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
†Infobright Inc.
ul. Krzywickiego 42/219, 02-078 Warsaw, Poland
{marcin.kowalski,dominik.slezak,piotr.synak}@infobright.com

***Abstract*—We discuss some enhancements of approximate SQL extensions available in Infobright's database technology. We explain how these new enhancements can speed up execution of complex correlated subqueries, which are quite popular in advanced database applications. We compare our research to the state-of-the-art solutions in the area of analytic databases. We also show in what sense our technology follows the principles of rough sets and granular computing.**

***Index Terms*—Analytic Databases, Data Granulation, Approximate SQL, Correlated Subqueries.**

I. Introduction

COLUMNAR databases provide a number of benefits with regard to both data storage (e.g.: data compression [1]) and data processing (e.g.: on-demand materialization [2]). 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 [3], leading toward modern analytic engines able to handle rapidly growing data sets.

Infobright's technology discussed in this article combines the benefits of columnar architectures with utilization of a metadata layer aimed at limiting data accesses while resolving queries [4]. In our solution, the content of each data column is split onto collections of values of some consecutive rows. Each data pack created this way is represented by its statistics. Such statistics are utilized by algorithms identifying data packs sufficient to complete particular stages of a given query execution [5].

The above solution is an example of a more general strategy of scaling complex computations on large data sets. Following the principles of rough sets [6] and granular computing [7], this methodology can be expressed by the four following steps: 1) Decompose data onto granules; 2) Create statistical snapshots for each of granules; 3) Do approximate computations on snapshots; 4) Whenever there is no other choice, access some of granules.

Certainly, such methodology should be compared to other approaches based on decomposing and merging computational tasks (see e.g. [8], [9]). It also requires addressing some details specific for particular applications, such as assuring sufficiently fast data decomposition methods, creating small but sufficiently informative snapshots, redesigning standard computational methods and accessing data granules in a minimized and optimized way.

Once the above challenges are solved, one can achieve a powerful framework where approximate computations assist execution of both standard and novel types of operations over massive data. In particular, in databases, it can be utilized to support both classical SQL statements and their approximate generalizations.

In the case of Infobright's RDBMS products, we distinguish three levels of such approximate assistance: 1) Optimization of data operations, which are components of execution of typical SQL statements [10]; 2) Computation of approximated results of SQL statements that occur in correlated subqueries in order to speed up answering to the main queries [11]; 3) Computation of approximated results visible to end users, which means actually extending standard SQL syntax [12].

In this paper we focus on the last two out of the above-outlined levels. We discuss an enhancement of our previous approximate SQL execution framework, now based on statistical metadata operations combined with accessing a small percentage of heuristically most informative data granules. We also report some performance tests proving that this new implementation can speed up a wide range of practically useful correlated subqueries.

The paper is organized as follows: Section II outlines some examples of understanding approximate SQL. Section III recalls our database technology, that is, Infobright's RDBMS solution. Section IV reports our previous research in the area of approximate SQL. Section V introduces new enhancements into our approximate SQL framework. Section VI recalls basic notions and challenges related to correlated subqueries. Section VII discusses application of enhanced approximate SQL framework to optimization of correlated subquery computation. Section VIII reports some performance tests corresponding to correlated subquery execution. Section IX discusses correlation between data sampling and precision of enhanced approximate query results. Finally, Section X concludes this study.

II. Approximate SQL

In such areas as, e.g., business intelligence or online analytics, there is a discussion whether the answers to SQL

978-1-4673-4471-5/$25.00

statements have to be always exact. As an example, in the case of rapidly growing and/or dynamically changing data sets, often with a limited/variable access or limited time/budget resuorces, the outcomes of a standard SQL statement may get practically useless prior to finishing its execution. An analogous dilemma arises for SQL-based versions of some machine learning methods, which work heuristically anyway [13].

There are many aspects of introducing approximate SQL. For instance, one can estimate actual SQL results by executing queries against data samples [14]. One can also rely on data synopses [15]. A database system may build numerous synopses for various subsets of columns and measures. Each query is then translated and calculated using only synopses. The answer obtained in such a way is returned as approximation. Yet another possibility is to generalize SQL operators in order to provide end users with more flexible answers [16]. Such extensions are especially useful when query constraints turn out to be too restrictive to produce any results or columns' data types are too complex for standard conditions and operations.

The two out of the above aspects can be clearly found in Infobright's technology. Our statistical metadata layer can be utilized to heuristically identify subsets of data packs that form sufficiently informative samples. It can be also interpreted as data synopses, which produces query approximations for both internal and external purposes. We go back to these aspects in Section IV, after getting more familiar with the considered database architecture. With regard to the third above-mentioned aspect, which is tending toward more flexible answers by modifying SQL operators, some of our approximate query functionalities can be regarded as SQL syntax extensions. However, one should remember that Infobright's main inspiration for query approximations is to speed up execution and/or decrease a size of standard SQL outcomes by answering with not fully accurate/complete results.

Let us also mention about one more important direction in the area of SQL approximations, specially related to the enhancements proposed in this paper. It is dedicated to controlling a complex query execution over time, by means of converging outcome approximations [17]. Such a convergence can take different forms, e.g.: monitoring partial query results until the calculation is completely finished, with possibility to stop it at any moment in time, or pre-defining some execution time and/or resource constraints that, when reached, will automatically stop further process even if the given query results are still inaccurate. We go back to this topic in Section VI.

III. Infobright's Architecture

In Infobright's RDBMS, rows loaded into a data table are partitioned onto so called row packs, each consisting of, by default, 2^{16} of rows. Each row pack is partitioned onto data packs, each consisting of 2^{16} values of a column. Thus, each data pack corresponds to a single row pack and a single data column. Data packs are compressed and stored on disk. During query execution selected data packs are read from disk, decompressed and analyzed. Decompression stage commonly constitutes query evaluation time or is its significant factor, so one tends to limit number of data packs decompressions. That is why, among others, some subset of most recently used data packs are available decompressed in memory. Prior to compression, various types of statistics are computed for each of data packs. For each data table, there is so called granulated information system with objects corresponding to row packs and attributes – to statistics. If justified we can consider also information systems with objects related to the pairs of row packs from different tables (see e.g. [5]). Granulated information systems constitute so called Infobright's knowledge grid.

There are various strategies of partitioning rows into row packs. In a general area of data processing and mining, we may refer to this task as to data granulation [7]. Appropriate data organization will have, among others, direct impact on informativeness of data synopses and – in consequence – on engine performance. However, we need to realize that in the case of database solutions expected to analyze large amounts of data being loaded in nearly real time such granulation needs to be very fast, possibly guided by some optimized criteria but utilized rather heuristically. While loading (or reloading, e.g., by an `insert from select` operation) data, one may control the number of rows in row packs (that is, the number of rows does not need to be always 2^{16}). To a certain extent, one may also slightly influence the ordering of rows for the purposes of producing better-compressed row packs described by more meaningful statistics, following analogies to data stream clustering [18].

Knowledge grid can be treated as a metadata layer. Besides simple statistics displayed in Fig. 1, it may also contain more advanced structures [19]. However, the size of all such structures needs to be far smaller than the size of data. Since granulated tables residing in knowledge grid are organized in a columnar way, so their columns, called knowledge nodes, can be selectively employed while querying. Another metadata layer stores information about location and status of data packs. It also contains lower-level statistics assisting in data decompression and, if applicable, translating incoming requests in order to resolve them using only partially decompressed data. There is also one more metadata layer responsible for interpretation of the contents of data packs and knowledge nodes. It contains, e.g., value dictionaries for columns with relatively small number of unique values and domain-specific descriptions of values of long alphanumeric columns, which may assist in their better compression [20].

The most fundamental way of using knowledge nodes during query execution refers to classification of data packs into three categories analogous to positive, negative, and boundary regions in the theory of rough sets [6]: Relevant

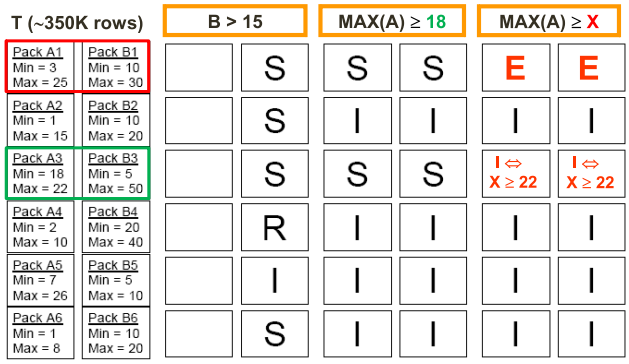

Fig. 1. Illustration for Section III. Simplified min/max knowledge nodes for numeric data columns `a` and `b` in table `T` are presented at the left side. Symbols R, S and I denote relevant, suspect and irrelevant data packs, respectively. E denotes a need of processing at the exact level. I/E means that the decision whether a given pack is irrelevant or requires exact processing will be made adaptively, depending on the outcomes of previous calculations [5].

```
select                  +--------+--------+------+
 min(a), sum(a), b      | min(a) | sum(a) |    b |
 from T                 +--------+--------+------+
 where b > 1            |      2 |      3 |    2 |
 group by b;            |      2 |      2 |    6 |
                        |   null |   null |    5 |
                        |      1 |      3 |    3 |
                        +--------+--------+------+

select roughly          +--------+--------+------+
 min(a), sum(a), b      | min(a) | sum(a) |    b |
 from T                 +--------+--------+------+
 where b > 1            |      1 |      2 |    2 |
 group by b;            |      2 |      3 |    6 |
                        +--------+--------+------+
```

Fig. 2. Example of SQL `select` statement and its rough version. Tables at the right sides display the results of both queries [10].

(R) packs with all elements relevant for further execution; Irrelevant (I) packs with no elements relevant for further execution; Suspect (S) packs that cannot be R/I-classified based on available knowledge nodes.

As an example, consider table `T` with 350,000 rows and columns `a` and `b`. We have six row packs: (A1,B1) for rows 1-65,536, (A2,B2) for rows 65,537-131,072 and so on. Consider knowledge nodes with minimum and maximum values displayed in Fig. 1. Assume there are no nulls and no other types of knowledge nodes available. Consider the following statement: `select max(a) from T where b>15;` [5]. The first execution stage uses min/max knowledge node for column `b` to classify data packs of `b` (and the whole corresponding row packs) onto relevant (B4), irrelevant (B5) and suspect (B1, B2, B3, B6) regions with respect to condition `B>15`. The second stage employs the third row pack to approximate the final result as `MAX(A)` $\geq$ `18`. Thus, only row packs (A1,B1) and (A3,B3) require further investigation. Maximum in A1 is higher than in A3, so, at the third stage, approximation is changed to `MAX(A)` $\geq$ `X`, where X depends on the result of exact processing of the first row pack. If X $\geq$ 22, then there is no need to access the third row pack. Otherwise, we need to proceed with its exact processing to get the precise outcome.

The first realistic implementation of an analogous granulated metadata support for query execution refers to [4], where the idea 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 statistics against `where` clauses in order to eliminate out-of-scope blocks. However, the above example shows that the analysis of fully in-scope row packs is equally useful, as in such cases it is enough to take statistics instead accessing data. Moreover, it is worth re-categorizing data packs as relevant/irrelevant iteratively, which distinguishes us from other approaches.

We refer to the literature for more examples how to utilize knowledge nodes in order to speed up data operations. Let us just mention that in case of multi-column `where` conditions some data packs, within a single row pack, can be identified as relevant/irrelevant while others may remain suspect. Furthermore, statistics and statuses of different data packs can be combined while processing complex (boolean, arithmetic, etc.) expressions treated as dynamically derived data columns [10]. (Actually, results of correlated subqueries investigated in further sections may be interpreted as such complex expressions.) It is also worth noting that knowledge nodes can be computed and efficiently used for intermediate results and structures created during query execution, such as e.g. hash tables storing partial outputs of joins and aggregations [21].

IV. Infobright's Query Approximations

Let us now go back to an outline of approximate SQL approaches that we developed so far. In [5], we introduced informally the notion of rough query, in order to better explain some internal data operations in our database engine. In [22], rough query was formalized for the purposes of correlated subquery optimizations. This topic was continued in [11]. Finally, in [12], we formulated the syntax of rough SQL statements and their results.

The following aspects of rough SQL were studied in [12]: 1) Query execution algorithms; 2) Internal results format; 3) Reporting results to end users. Let us explain those aspects using an example from Section III. Recall that by utilizing min/max knowledge nodes of columns `a` and `b` we could compute that the outcome of `select max(a) from T where b>15;` is at least 18. This is because of row pack (A3,B3), within which there must be at least one row satisfying condition `b>15` and having value not less than 18 on `a`. We obtain approximation in the form of interval $\langle 18, 25\rangle$. This would be actually the answer to query `select roughly max(a) from T where b>15;` within the rough SQL framework introduced in [12].

Every `select` SQL statement returns a set of tuples

labeled with the values of some attributes corresponding to the items after `select`. Approximation of a query answer can be specified as statistics describing attributes of such a tabular outcome. Rough SQL can be assumed to produce a kind of knowledge grid for such attributes. Accordingly, the results of rough queries in [12] were provided as ranges $\langle lower, upper\rangle$ approximating the results of the corresponding standard queries. Here we should notice that, as for now, our approach concerns only numeric columns, however some efforts of analogous representation for alphanumeric columns have been done too.

Consider an example in Fig. 2, where the task is to compute aggregations `min(a)` and `sum(a)` with respect to column `b`. The resulting tuples correspond to the groups induced by `b`. There are three attributes: `min(a)`, `sum(a)` and `b`. Rough version of the same SQL computes ranges for two aggregations and the grouping column. Rough outcome tells us that for each resulting tuple, if its value of `min(a)` is not `null`, then it is for sure between 1 and 2. Similarly, `sum(a)` is in $\langle 2, 3\rangle$, and `b` is in $\langle 2, 6\rangle$.

In parallel to our research on rough SQL, in [19] we started to extend Infobright's framework toward a kind of inexact querying. Those studies were further continued in [23]. In this approach, we do not compute query result approximations, neither with full nor partial confidence that they are correct. Instead, we attempt to speed up the query execution process by quickly producing potentially inaccurate results that look in a standard way.

In this case, we follow an expectation that approximate queries yield tuples being almost the same as those resulting from standard queries. We enriched knowledge nodes and the corresponding query processing functions in order to compute degrees of data packs' (ir)relevance. We also investigated inexact knowledge nodes that describe data packs almost correctly, neglecting local outliers in order to provide crisper min/max intervals. Then, basing on some analogies with extensions of rough sets (see e.g. [24]), we loosened the criteria for R/I pack status, i.e., we treated almost not suspect packs as if they were truly (ir)relevant . This way, we created a framework for computing inexact query outputs with a decreased need for data access.

In [23], we also proposed a more probabilistic approach, where so called degrees of (ir)relevance, i.e., dynamically derived coefficients that heuristically estimate how many elements of particular data packs might satisfy query conditions, are employed to randomly select row packs for further processing. In this case, we can work with standard knowledge nodes [5]. However, with some probability proportional to the above degrees, data packs that are potentially almost relevant or irrelevant may be classified as fully relevant or irrelevant, respectively.

The above randomized approach inspired us to develop a number of intelligent sampling techniques. Given the architecture described in Section III, it is important to randomly select a reasonably small subset of row packs and then choose a sample of rows from those row packs only. A challenge is to select row packs providing sufficient representativeness of the final sample. In this case, utilizing knowledge nodes in order to compute degrees of row packs' (ir)relevance with respect to particular query conditions is very helpful. While producing a sample, fully relevant data packs should have the highest chance to get selected while fully irrelevant data packs should not be considered at all. For suspect data packs, their degrees of (ir)relevance should directly influence the probability of taking them into account during sample generation. In further sections, we refer to queries executed over samples generated in this way as to sampled queries.

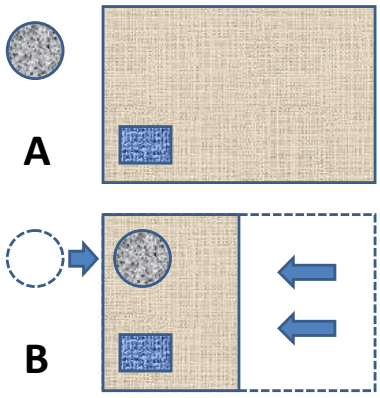

Fig. 3. A: Comparison of a result of a standard `select` statement (symbolized by smaller rectangle) with the results of its sampled (circle) and rough (bigger rectangle) versions; B: Illustration how a sampled result can be modified using knowledge nodes and how rough result may become crisper by accessing some of data packs.

V. Enhanced Rough SQL

Rough SQL reported in [12] provides fast responses, but the spans of approximations are often not informative enough to end users nor internal optimization mechanisms. On the other hand, sampled queries discussed in the end of Section IV provide fast responses (although not as fast as rough queries) that are potentially more informative, but end users have no means to analyze how accurate they are (see Fig. 3A). Certainly, it is crucial to approximate errors occurring for particular data packs and to propagate them through the whole query execution process in order to provide end users with overall estimation. This task gets more complicated along a growing complexity of analytical SQL statements and needs to be considered for all query methods. Thus, a question arises whether it is possible to combine benefits of rough queries and sampled queries within a unified framework.

The above question can be answered in at least two ways (see Fig. 3B). One of possibilities is to enrich sampled queries with an analysis whether their results remain within the bounds produced by rough versions of the same queries. If a given sampled result is not within such bounds, we can quickly move it toward the closest *edge* of rough SQL approximation. Another possibility is to

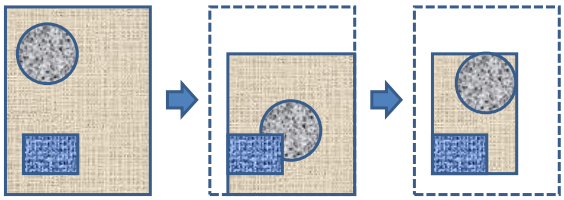

Fig. 4. Illustration how enhanced sampled query results (circles) and enhanced rough query results (bigger rectangles) are expected to behave with respect to a standard query result (smaller rectangle) while accessing more and more data packs.

develop a crisper version of rough SQL that would provide better ranges for both sample-based and fully exact querying, by combining statistical information provided by knowledge nodes with crisp information about opened data packs. Actually, in this case one should expect that rough SQL enriched by additional information about the content of a subset of suspect data packs should provide reasonable ranges for sampled SQL, where a sample is created basing on the same subset of data packs.

From a technical point of view, sampled querying is based on selecting subsets of row packs and then selecting some rows from those row packs. Therefore, it requires decompression of those row packs. Once we have some row packs decompressed, their content can be used to improve rough SQL results in the same time. Realization of this idea is actually very simple: When a data pack is accessed, we can replace its statistics stored in Infobright's knowledge grid with very crisp information about only those of its elements, which are useful at a given stage of query execution (e.g.: which correspond to rows satisfying conditions over other columns in a given `select` statement). This means that knowledge nodes, or rather their copies utilized while executing a given query, can become crisper over time, leading to a crisper final result.

Let us emphasize that from a practical point of view there is no requirement for results of sampled queries to fall within the bounds provided by their rough counterparts. It depends entirely on expectations of end users. For distinction, we will refer to a sampled query mechanism enriched by rough results analysis as to an enhanced sampled querying. Analogously, we refer to a rough query mechanism enriched by information from a subset of suspect data packs as to an enhanced rough querying.

Let us go back for a while to the idea presented in [12], where rough SQL was based entirely on Infobright's knowledge grid. We implemented a number of techniques utilizing knowledge nodes at particular stages of execution of `select` statements. All of them are based on heuristics analogous to the mechanisms of dynamic approximation of standard query outcomes, as shown in Section III. Approximations are often not perfectly precise but can be obtained very fast. However, additional data access may not necessarily lead to a dramatic slowdown of rough SQL execution. For instance, it is worth noting that Infobright caches recently used data packs in memory. Integration of information residing in knowledge nodes with such packs within a framework for enhanced rough SQL may significantly improve precision of results.

Let us also recall that Infobright's query execution should be treated as an iterative process, where knowledge nodes and information acquired from data accessed at a particular stage can be employed for heuristic selection of the very next packs that are likely to mostly contribute to narrowing down the $\langle lower, upper \rangle$ ranges. In general, we can think about several strategies of choosing data packs or row packs to be processed. For instance, to improve enhanced rough SQL results with minimum data accesses, we should choose packs that are expected to maximize information gain for after-select attributes (which may follow exactly the same heuristics as in the example in Section III). On the other hand, in order to increase reliability of sample-based querying, we should rather choose row packs that seem to be statistically representative with respect to the query domain (as discussed in the end of Section IV). Finally, in order to speed up execution without losing quality, we should attempt to combine the above strategies with information about data packs that are currently cached in memory.

Finally, let us note that enhanced rough query result ranges become crisper (or at least not less crisp) when more row packs are taken into account. There is no guarantee that a distance between actual query and approximated query results decreases monotonically. However, while enhanced rough query result ranges become crisper, we intuitively expect to approximate the above distance better and better (see Fig. 4). As mentioned in Section II, there are also some approaches where an end user provides an upper bound for query processing time and acceptable nature of answers (partial or approximate). One can imagine an analogous framework designed for Infobright, wherein a query is executed starting with rough information and then it is gradually refined by decompressing heuristically selected pieces of data. The execution process can be then bounded by means of various parameters, such as time, acceptable errors, or percentage of data accessed. We can also consider a design of Infobright-specific incremental querying, although it would require further extensions of end user interfaces.

VI. Correlated Subqueries

A correlated subquery is 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. For illustrative purposes, consider the following example of the `where` clause of the outer query: `U.x=(select max(T.a) from T where T.b>U.y)`. If `U` and `T` are large, then the execution of the outer query on `T` may be time consuming. However, as pointed out in [22], one can quickly derive rough answers

to particular subqueries and, for each row in `T`, check whether the above condition could be successfully resolved by using such dynamically obtained statistics.

Importance of correlated subqueries 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.

Internal rough query algorithms have been utilized in Infobright's RDBMS for a longer time in order to speed up correlated subqueries [11]. Surely, one can adopt here some well-known methods of the correlated subquery optimization [25], such as caching results for last-observed parameters or rewriting nested queries into semi-joins. However, in some cases, the nested part of a query may still need to be processed in its direct form for a huge amount of rows of an external table. Fast production of query approximations may eliminate the need of a standard computation at least for a subset of such rows.

For a subquery in the `where` clause, we launch its rough version with parameters induced by each consecutive row in the outer table. We attempt to use its rough outcome to avoid the exact mode of execution. For the above-mentioned example of the clause `U.x=(select max(T.a) from T where T.b>U.y)`, we can express it in two stages: $(x, \widetilde{s}(y))$ and $(x, s(y))$. The first of them symbolizes, for a given row, the comparison of its value on column x with a rough result of the subquery parameterized by its value on column y. If such a comparison is not sufficient to decide whether that row satisfies the `where` clause, i.e., if the value of `U.x` does not yield fully relevant or fully irrelevant condition when comparing with the result of statement `select roughly max(T.a) from T where T.b>U.y;` then, for that particular row, we need to proceed with the standard subquery execution $s(y)$.

A broader roadmap for rough query-related future optimizations of correlated subqueries was presented in [11]. In all cases presented in that paper, the preciseness of rough query outcomes is crucial for ability to minimize data access and speed up computations.

VII. Optimization of Correlated Subqueries by Enhanced Rough SQL

Efficiency of different forms of approximate or randomized query frameworks can be examined in multiple ways. For example, in [23] we investigated stability and reliability of the results of top-k queries while tuning parameters of random selection of almost suspect data packs. As another example, in [12] we presented some strategies utilizing rough queries to speed up standard SQL calculations. In this paper, we focus on the original idea of using rough SQL to improve performance of correlated subqueries [22]. Our goal is to show that additional employment of enhanced rough SQL mechanisms described in Section VI can lead to further gains in this area. In the nearest future, we plan to extend our tests also onto other aspects of practically most promising enhanced rough SQL applications.

TABLE I
The amount of data packs accessed while querying. The TPCH100 query is a standard benchmark query. The TEST_1 and TEST_2 queries come from Infobright's internal benchmark framework. The SYNAT_1 and SYNAT_2 queries are taken from the SYNAT project [26]. The remaining queries represent Infobright's customers.

SQL statement	rough → exact	rough → enhanced → exact
TPCH100	2869154	185330
TELCO_1	21663	12087
TELCO_2	9579	3
WEBLOGS	51047	28769
TEST_1	40707	40707
TEST_2	40707	3
SYNAT_1	333582	333552
SYNAT_2	252868	251242

At the first glance, one may suspect that accessing data packs slows down producing rough subquery outcomes. However, crisper outcomes let us eliminate more calculations at a higher level. Moreover, we implemented it in such a way that the enhanced rough query execution is triggered only if standard rough query fails. In our previous implementation, we would follow with exact subquery immediately. In the new implementation, we proceed with exact computation only if enhanced rough query fails to give us results that would be precise enough. Therefore, it is not about a trade-off between the amount of accessed packs and precision of results – the way it was implemented assures that we always win by applying this new mechanism. Symbolically, we replace the previous strategy expressed by the pair of stages $(x, \widetilde{s}(y))$ and $(x, s(y))$ by a triple, where $(x, \widetilde{s}(y))$ is followed by $(x, \overline{s}(y))$ and then by $(x, s(y))$, where $\overline{s}(y)$ denotes an enhanced rough execution of inner query s with parameters set up by a value of outer column y. Surely, the other strategies reported in [11] could be extended this way as well.

Mechanism of enhancing is parameterizable. Before running query we can choose how many row packs for single subquery evaluation are permitted to be additionally decompressed. If not specified explicitly, all experiments described in the following section were performed for the only one row pack per subquery evaluation permitted.

VIII. Experimental Results

Table I illustrates one of specific classes of correlated subqueries aimed at selecting and processing extreme cases for some categories. Such nested queries differ from

relatively simpler `group by` statements in a way of considering observations marked as extreme. They are useful for some aspects of time-oriented analysis, extreme behaviour or outliers tracing. Specific tasks corresponding to such queries may look, e.g., as selecting all latest news about each company in the news table

```
select * from news n1 where pub_date
  = ( select max(pub_date) from news n2
      where n1.company_id = n1.company_id )
```

or, for a set of stemmed documents, selecting all stems that are of a maximal TF-IDF value for particular documents (see [26] for comparison)

```
select stem from stemmed_docs x where tf_idf
  = ( select max(tf_idf) from stemmed_docs
      where x.id_doc = id_doc )
```

The results in Table I are displayed by means of data packs that had to be opened. Such measure for considered class of queries, reflects real performance of query evaluation fairy accurately. We will then identify number of decompressed data packs with query performance understood as time spent waiting for the results.

The last column reports the results of our new approach, where the intermediate enhanced rough SQL version of a correlated subquery is computed with only one heuristically selected row pack to be additionally accessed. This shows a huge potential of the proposed method. For some of the considered queries, we were able to provide and test equivalent non-nested `select` statements with additional `group by` or `self-join` operators. Usually, such rewriting significantly improves performance. However, for queries such as TELCO_2 and TEST_2, it was impossible to find an alternative way of execution that would be comparably fast to the strategy based on enhanced rough SQL.

IX. Discussion

Let us now elaborate more on how the data sampling intensity correlates with the precision of enhanced rough query results. First of all, it turns out that queries with correlated subqueries based on MIN/MAX aggregations can particularly gain from enhanced rough SQL as such aggregations are sensitive to information about single observations. Figure 5 illustrates it by presenting enhanced rough queries for selected values of additional decompressions of row packs per query evaluation. Parameter equal to 0 means an ordinary rough SQL (with no extra data accesses). We can see that after accessing nine additional row packs we achieve fully crisp result.

On the contrary, the precision of, e.g., COUNT is expected to be linear with respect to the number of additional accesses [23]. It makes the enhanced rough query sampling strategy for COUNT (as well as SUM and AVG) questionable, although in some cases accessing several data packs may still help in resolving a query condition. Generally, these types of aggregations may be handled better by enhanced sampled queries, although inexactness of their outcomes is not always acceptable by end users.

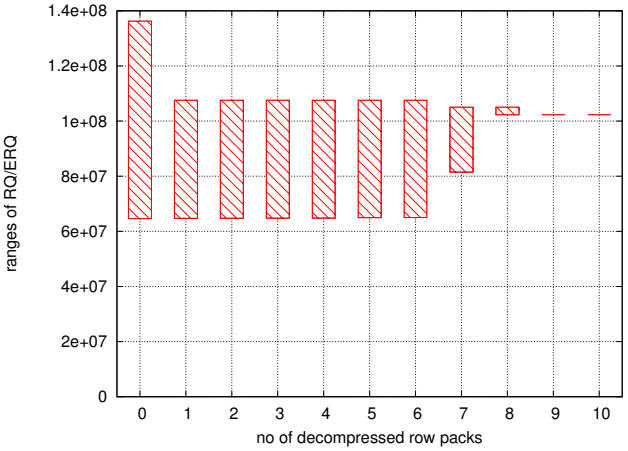

Fig. 5. Exemplary scenario of influence of increasing number of row pack decompressed during enhanced rough query (ERQ) calculation on ranges returned by ERQ performed on customer's data (SELECT ROUGHLY MIN(col) FROM t WHERE cond); 0 stands for original rough query; for 9 row packs we achieve crisp result

It is also worth elaborating on sampling strategies that could decrease a need of additional decompressions by, e.g., achieving crisp results after shorter period of time. For example, one of such strategies for MIN/MAX aggregations is to start decompression from data packs with the most extreme values represented by their knowledge nodes (like in Figure 5). A similar strategy could be followed for COUNT DISTINCT aggregations by referring to data packs with the highest estimated number of distinct values. However, analyzing results from Figure 5 (in particular the little improvement in range estimation between cases with parameter set to 1 and 6) we anticipate the need of enrich sampling strategy development of utilizing information contained e.g. in histograms.

Additional decompressions may be done randomly but a more intelligent procedure is recommended. The presented results also suggest that in case of correlated subquery optimization such mechanism may be complementary to syntactic reformulation of queries (for instance into JOINs). It should be also pointed out that some on-load row reorganization strategies aimed at improving the quality of knowledge nodes may further increase effectiveness of sampling mechanisms [18]. This is because, generally, the increase in the quality of rough information represented by knowledge nodes helps a lot in all aspects of Infobright's querying algorithms.

X. Conclusions

We discussed some enhancements of rough SQL framework developed by Infobright. We reported how such enhancements can speed up execution of correlated subqueries. In our future research, we are going to examine

further cases of practical usage of enhanced rough SQL. We will also attempt to better analyze convergence of query approximations depending on various strategies of sampling blocks of rows during query execution.

References

[1] P. White and C. French, "Database System with Methodology for Storing a Database Table by Vertically Partitioning all Columns of the Table," US Patent 5,794,229, 1998.

[2] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden, "Materialization Strategies in a Column-Oriented DBMS," in *ICDE*, 2007, pp. 466–475.

[3] P. A. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-Pipelining Query Execution," in *CIDR*, 2005, pp. 225–237.

[4] J. K. Metzger, B. M. Zane, and F. D. Hinshaw, "Limiting Scans of Loosely Ordered and/or Grouped Relations Using Nearly Ordered Maps," US Patent 6,973,452, 2005.

[5] D. Ślęzak, J. Wróblewski, V. Eastwood, and P. Synak, "Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1337–1345, 2008.

[6] Z. Pawlak and A. Skowron, "Rudiments of Rough Sets," *Information Sciences*, vol. 177, no. 1, pp. 3–27, 2007.

[7] A. Bargiela and W. Pedrycz, *Granular Computing – An Introduction*. Kluwer Academic Publishers, 2002.

[8] Y. Bu, B. Howe, M. Balazińska, and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 285–296, 2010.

[9] M. Szczuka and D. Ślęzak, "How Deep Data Becomes Big Data," in *IFSA-NAFIPS*, 2013.

[10] D. Ślęzak, P. Synak, J. Wróblewski, J. Borkowski, and G. Toppin, "Rough Optimizations of Complex Expressions in Infobright's RDBMS," in *RSCTC*, 2012, pp. 94–99.

[11] D. Ślęzak, P. Synak, J. Borkowski, J. Wróblewski, and G. Toppin, "A Rough-columnar RDBMS Engine – A Case Study of Correlated Subqueries," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 34–39, 2012.

[12] D. Ślęzak, P. Synak, G. Toppin, J. Wróblewski, and J. Borkowski, "Rough SQL – Semantics and Execution," in *IPMU, vol. 2*, 2012, pp. 570–579.

[13] J. Wróblewski, "Analyzing Relational Databases Using Rough Set Based Methods," in *IPMU, vol. 1*, 2000, pp. 256–262.

[14] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized Stratified Sampling for Approximate Query Processing," *ACM Trans. Database Syst.*, vol. 32, no. 2, p. 9, 2007.

[15] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim, "Approximate Query Processing Using Wavelets," *VLDB J.*, vol. 10, no. 2-3, pp. 199–223, 2001.

[16] S. Zadrożny and J. Kacprzyk, "Issues in the Practical Use of the OWA Operators in Fuzzy Querying," *J. Intell. Inf. Syst.*, vol. 33, no. 3, pp. 307–325, 2009.

[17] S. Chaudhuri, V. R. Narasayya, and R. Ramamurthy, "Estimating Progress of Long Running SQL Queries," in *SIGMOD*, 2004, pp. 803–814.

[18] D. Ślęzak, M. Kowalski, V. Eastwood, and J. Wróblewski, "Methods and Systems for Database Organization," US Patent 8,266,147 B2, 2012.

[19] D. Ślęzak and V. Eastwood, "Data Warehouse Technology by Infobright," in *SIGMOD*, 2009, pp. 841–846.

[20] M. Kowalski, D. Ślęzak, G. Toppin, and A. Wojna, "Injecting Domain Knowledge into RDBMS – Compression of Alphanumeric Data Attributes," in *ISMIS*, 2011, pp. 386–395.

[21] D. Ślęzak, P. Synak, J. Wróblewski, and G. Toppin, "Infobright Analytic Database Engine Using Rough Sets and Granular Computing," in *IEEE GrC*, 2010, pp. 432–437.

[22] P. Synak, "Rough Set Approach to Optimisation of Subquery Execution in Infobright Data Warehouse," in *SCKT (PRICAI Workshop)*, 2008.

[23] D. Ślęzak and M. Kowalski, "Towards Approximate SQL – Infobright's Approach," in *RSCTC*, 2010, pp. 630–639.

[24] W. Ziarko, "Probabilistic Approach to Rough Sets," *Int. J. Approx. Reasoning*, vol. 49, no. 2, pp. 272–284, 2008.

[25] M. Elhemali, C. A. Galindo-Legaria, T. Grabs, and M. Joshi, "Execution Strategies for SQL Subqueries," in *SIGMOD*, 2007, pp. 993–1004.

[26] D. Ślęzak, K. Stencel, and H. S. Nguyen, "(No)SQL Platform for Scalable Semantic Processing of Fast Growing Document Repositories," *ERCIM News*, vol. 2012, no. 90, 2012.