Rough SQL – Semantics and Execution

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

[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,toppin,jakubw,januszb}@infobright.com

Abstract. We introduce *rough query*, which is a new approach to defining and computing SQL approximations. Rough query results are reported by means of simple metadata of attributes in an information system that would be a result of a standard `select` statement. The proposed approach is already available as an SQL extension in both Infobright Community and Enterprise Editions. Rough queries are computed practically in real time by basing on the statistical information layer, which was used so far only for the standard query optimization.

Keywords: Analytic RDBMS Solutions, Approximate SQL, Rough SQL.

1 Introduction

In such areas as business intelligence and web analytics there is an ongoing debate whether the answers to SQL statements have to exact in all cases. Of special importance are analytic sessions of advanced users who explore the data using various types of telescoping queries (where the form of each of queries depends on the results of the previous ones), trying to look around the data prior to specifying the final requests. The same question occurs in a case of SQL-based machine learning algorithms, which are often based on heuristics, randomness and inexactness anyway [1,2].

Approximate SQL can be understood in many ways. One can, e.g., generalize SQL operators to provide users with more flexible answers [3,4]. Another way is to keep the standard meaning of SQL but speed up an execution or decrease the size of outcomes by answering with not fully accurate/complete results. Motivation for SQL approximations may be related to complexity of queries and data sources, dynamically changing data with a limited access (e.g., for sensory data and data streams), but also to huge data sets for which there is a need to monitor convergence of a query execution over time, regardless of whether the final answers are to be standard or approximate [5,6].

In this paper, we discuss *rough query* (or *rough SQL*), which is an extension of the standard SQL available in the Infobright Community and Enterprise Editions (abbreviated as ICE and IEE, respectively[1]). Query results are approximated by using Infobright's *knowledge grid* (statistical summaries of horizontally and vertically partitioned

[1] `www.infobright.org`; `www.infobright.com`

S. Greco et al. (Eds.): IPMU 2012, Part II, CCIS 298, pp. 570–579, 2012.

data pieces). Let us treat an answer to a `select` statement as an information system [7,8] with attributes corresponding to the items after `select` (e.g.: `a` and `count(*)` in `select a, count(*) from T group by a;`) and objects corresponding to the result's tuples (e.g.: distinct values of `a`). Approximation of an answer is specified as metadata of such an information system. Thus, rough query produces a kind of a knowledge grid of the query result by using the input metadata.

Rough SQL execution is orders of magnitude faster than in the standard case. Users obtain approximate results of all types of `select` statements practically in real time. In the current implementation, rough queries are free from *false positives*, i.e., their results are always guaranteed to approximate outputs of the corresponding standard queries in a valid way. This leads to a number of interesting applications. In particular, Infobright's users can improve the current knowledge-grid-based execution of standard SQL [9,10] by building their own scripts involving rough SQL.

The paper is organized as follows. Section 2 outlines the related work. Section 3 recalls Infobright's architecture. In Section 4, we introduce rough SQL and discuss examples of its use in practice. Section 5 categorizes possible future extensions of the current rough query's implementation. Section 6 concludes the paper.

2 Related Work

The proposed rough SQL is a specific example of an approximate query framework, as it supports users with fast approximate answers to `select` statements. Therefore, let us outline some relevant aspects of the research on approximate querying.

First of all, there are techniques based on estimating actual answers by executing queries against data samples [11]. Another trend is to rely on various types of data summaries [12]. One may interpret Infobright's knowledge grid as a kind of data summary layer, although it is used to support the query execution more deeply than in other RDBMS technologies. In the future, our knowledge grid may be also applied for efficient identification of data pieces that form statistically representative data samples.

In [3], the answer to a `select *` query is approximated as not bigger or not smaller than the standard one, dependent on whether lower or upper rough set approximations [8] are in use. This idea is partially analogous to rough query proposed in this paper, although the underlying algorithmic framework is different. Also, the ability to use the approach proposed in [3] for large data sets is limited, and semantics of approximate results is not provided for all types of `select` statements.

In [6], a framework for time-constrained SQL is proposed, where users can specify an upper bound for a query processing time and an acceptable nature of answers. Similar idea was presented earlier in [5]. An analogous framework can be designed in the future for Infobright by approximating a query result basing on the knowledge grid and then refining it gradually by accessing heuristically selected data pieces.

In [13], we investigate how to extend Infobright's knowledge grid in order to reduce the required data access, even if it leads to slightly inexact outcomes. Unfortunately, the proposed method requires tuning of a number of parameters, which is a general problem of approximate SQL methods. Another issue with our approach reported in [13] is an insufficiently clear interpretation of approximate answers to arbitrary `select` statements. Rough SQL proposed in this paper is far more straightforward, although further

research on practically useful semantics of query results is still needed. With this in mind, one may seek for inspiration also beyond the database solutions [14].

3 Infobright's Architecture

Infobright's RDBMS is based on the principles of data compression and columnar storage that are widely known in database research and industry [15], in combination with the elements of rough sets and rough computing [8] that provide fast execution of unexpected analytic SQL statements over huge amounts of data.

Rows loaded into a data table are partitioned onto *blocks*, each consisting of 2^{16} of rows. Each block is partitioned onto *packs*, each consisting of 2^{16} values of a column. Various types of statistics are computed for each of the packs. Finally, the packs are compressed and written on a disk. For each data table, there is a *rough information system* with objects corresponding to blocks and attributes corresponding to the above-mentioned statistics. Such systems are stored in Infobright's knowledge grid.

In [9], we showed how to use the knowledge grid to speed up various types of data operations. Let us recall how it is applied in order to classify packs into three categories that are analogous to *positive*, *negative*, and *boundary regions* known from the theory of rough sets: *Relevant (R) packs* with all data elements relevant for further execution; *Irrelevant (I) packs* with no data elements relevant for further execution; *Suspect (S) packs* that cannot be R/I-classified based on the knowledge grid.

Also in [9], we presented the following case study: Consider table `T` with 350,000 rows and columns `a` and `b`. We have six blocks: (A1,B1) corresponds to rows 1-65,536, (A2,B2) – to rows 65,537-131,072, etc., until (A6,B6) corresponding to rows 327,681-350,000. Consider an example of *min/max statistics* that contain the minimum and maximum values for each separate pack, as displayed in Figure 1a. For simplicity, assume there are no nulls in `T` and ignore all other types of statistics gathered in Infobright's knowledge grid. The SQL statement of interest is as follows:

```
select max(a) from T where b > 15;
```

According to the min/max statistics for the column `b`, packs B1, B2, B3, B6 are S, B4 is R, and B5 is I (Figure 1b). According to the min/max statistics for the column `a`, we obtain the following approximation: `max(a)` subject to `b > 15` is between 18 and 25. Thus, after re-classifying packs and their corresponding blocks, only (A1,B1) and (A3,B3) require further investigation (Figure 1c). The maximum in A1 is higher than in A3. Thus, (A1,B1) is the first one to be processed. Depending on the analysis of its rows, (A3,B3) will become I or will need the exact processing too (Figure 1d).

Figure 1 shows that one can interpret our query execution as an approximation process. It begins with the knowledge grid and then gradually steps down to the level of exact data. In the above example, the first approximation equals to $\langle 18, 25 \rangle$, and the second one would be obtained after decompressing (A1,B1). This also illustrates how rough set regions (corresponding to the R/S/I pack statuses) can change over time. In general, Infobright's approach to the query execution may be an interesting practical model for researchers studying the theory of rough sets (see also Section 5.3).

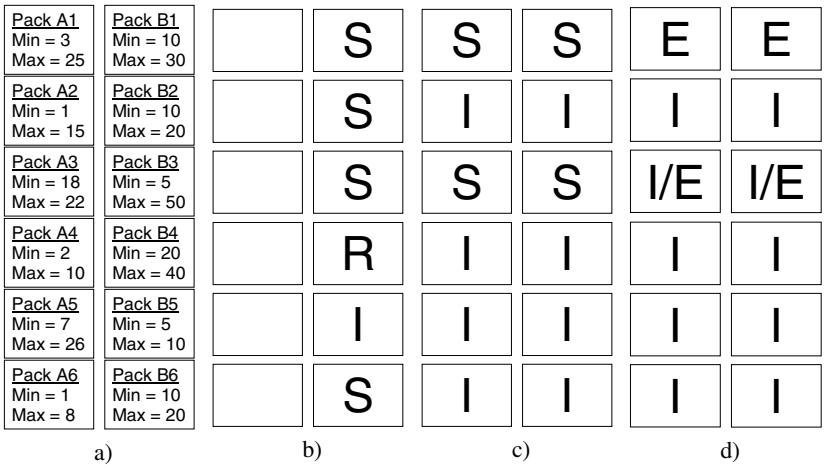

Fig. 1. An illustration for Section 3: (a) Min/max statistics; (b,c,d) Execution stages. R, S and I denote Relevant, Suspect and Irrelevant packs, respectively. E means the need of exact processing.

4 Rough Query

4.1 Current Layout

In its current implementation, rough query's result is given as a range $\langle lower, upper \rangle$ approximating a result of the corresponding standard query. Every standard `select` statement returns a set of tuples labeled with the values of some attributes – a kind of an information system, as noted in Section 1. Consider the following example:

```
select min(a), sum(a), b
  from T where b > 1 group by b;
```

```
+--------+--------+------+
| min(a) | sum(a) |    b |
+--------+--------+------+
|      2 |      3 |    2 |
|      2 |      2 |    6 |
|   null |   null |    5 |
|      1 |      3 |    3 |
+--------+--------+------+
```

The above query computes aggregations `min(a)` and `sum(a)` with respect to the column `b`. The resulting tuples correspond to the groups induced by `b` and there are three attributes: `min(a)`, `sum(a)` and `b`. Now, consider an analogous rough query:

```
select roughly min(a), sum(a), b
  from T where b > 1 group by b;
```

```
+--------+--------+------+
| min(a) | sum(a) |    b |
+--------+--------+------+
|      1 |      2 |    2 |
|      2 |      3 |    6 |
+--------+--------+------+
```

Such query computes ranges for two aggregations and the grouping column. Query's 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 $\langle 2, 6 \rangle$.

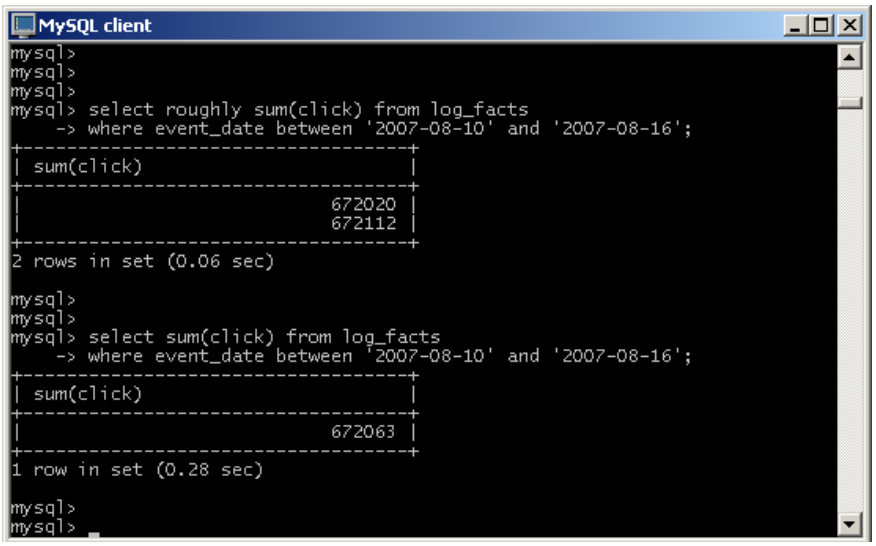

Fig. 2. Two versions of the same query executed on a standard PC against a table `log_facts` with 15 billion rows. The rough query runs in 0.06 sec. An analogous standard query runs in 0.28 sec. A usage of the MySQL client is possible thanks to leveraging the MySQL framework for storage engines (with some changes related to Infobright-specific query optimization [9]).

4.2 Solution Components

The architecture of rough SQL may be considered by means of the following layers: 1) the query execution algorithms, 2) the internal format of result, and 3) the methods that interpret, report, aggregate, and visualize that result.

The algorithms executing rough queries are based entirely on Infobright's knowledge grid, without accessing any packs of exact data. They do not construct explicitly the standard result as an intermediate stage. We implemented a number of techniques specific for particular stages of an 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 3. The speed of rough SQL implemented this way is illustrated by Figure 2. The obtained approximations are often not perfectly precise, e.g., it can happen that $\langle lower, upper \rangle$ for `sum(a)` in the example from Section 4.1 is $\langle 0, 4 \rangle$. We continue our work on improving this kind of precision.

The internal rough query algorithms have been used in our RDBMS for a longer time in order to speed up correlated subqueries [10]. Surely, one can adopt here some well-known methods of the correlated subquery optimization. However, the nested part of a query may still need to be processed 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 instance, for the `select` statement from Figure 3, such a standard computation is avoided, if `clicks + 5000` is less than *lower* of `(select ...)` for a given row of `user_campaign`.

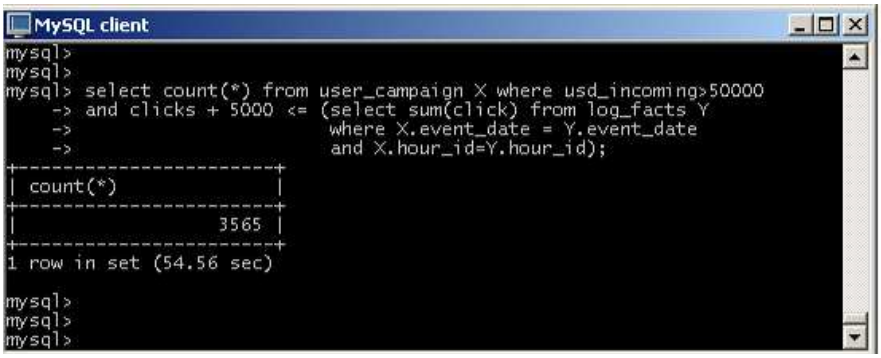

Fig. 3. An analytic SQL with a correlated subquery. It runs in 54.56 sec for the same data (and on the same machine) as in the case of Figure 2. With internal rough-query-based optimizations of correlated subqueries [10] turned off, the execution time increases to 1 min 47 sec.

There are some analogies between rough SQL introduced in Section 4.1 and the above rough optimization of correlated subqueries. In both cases, the aim is to increase query performance and to avoid already mentioned false positives, i.e., the situations when the $\langle lower, upper \rangle$ intervals are not valid approximations of the actual results. In the latter case the reason is obvious: invalid approximations might lead to invalid results of `select` statements. With regard to the former case, we noticed that fully valid approximations are, at least for now, easier to use by our customers.

A significant difference between externally exposed rough SQL and the methods hidden inside the database engine is the requirement of a comprehensive semantic of the output. As claimed in Section 1, a rough query result might take a form of the knowledge grid describing an outcome of the corresponding standard statement. However, the statistical structures present in Infobright's knowledge grid would be totally unclear for users. This is why we postulate separation of two already-mentioned layers related to the internal output format and its external simplified interpretation. Our ultimate goal is to have the same algorithms and internal output formats for all kinds of a rough query usage, with a difference at the level of external result delivery.

In the current implementation, for the sake of clarity, we simplified not only the semantics of rough query results but also the rough query syntax. In the future, one may enrich it with additional parameters responsible, e.g., for a tradeoff between the speed and precision of execution. We discuss it in more details in Section 5.

4.3 Usage Examples

Our main motivation for introducing rough SQL is to run queries in (nearly) real time, as stated in Section 1. Rough queries can be executed, e.g., in order to narrow down the search space before executing regular SQL statements. This way, the time-consuming analytic queries become better focused thanks to the intermediate procedures, which are based on faster rough SQL. Such an approach, further referred to as the telescoping, can be useful in investigative analytics and data exploration.

Rough queries can be also used in operational business intelligence, where heavy aggregates or projections are executed with ad-hoc filters. For instance, in telecommunication applications, often the goal is to troubleshoot issues in the field and determine the trends of errors. The bulk of the corresponding queries generated by support engineers can return no results, and yet they are executed each time over huge data sets. By running rough SQL, one can anticipate such cases, reducing both, the time to arrive at an answer and the overhead associated with the speculative queries.

Another application of rough SQL relates to using Infobright as a pre-staging repository aimed at producing data sets for further analysis. Queries leading toward the required data sets can be very time-consuming as well. Moreover, prior to query execution, there is no guarantee that the outcome is going to satisfy given analytical requirements. In this scenario, rough SQL results can quickly provide descriptions of output data sets prior to their materialization. Basing on such descriptions, users can decide whether the underlying queries need further modifications.

The execution speed may not be the only practically meaningful advantage of rough SQL. Let us note that result sizes of rough queries are often orders of magnitude smaller than in the case of their corresponding standard `select` statements. It may be important, e.g., if an outcome of a complex multi-column `group by` is to be sent to users or third-party applications. Thus, when discussing possible extensions in Section 5, we should remember about a compactness of rough query outputs.

The last use case reported in this section may be a bit surprising as it relates to a usage of rough SQL in order to improve performance of standard queries. Let us consider the following type of analytic statements as an example:

```
select distinct a from T where F order by a desc limit L;
```

Let us refer to the above query as to $Q1$. Assume that `a` is a numeric column of `T`, `L` is a positive integer and filter `F` may be any combination of conditions, which often happens in an ad-hoc exploratory querying. Consider the following $Q2$:

```
select distinct a from T
               where F and a > X order by a desc limit L;
```

Our task is to come up with a procedure of a heuristic search for a threshold `X` such that $Q2$ leads to the same outcome as $Q1$, by itself or in combination with procedure's intermediate results. $Q2$ is supposed to execute faster than $Q1$. Also, execution of the search for `X` should be significantly faster than in the case of both $Q1$ and $Q2$.

Algorithm 1 illustrates how to use rough SQL not only to find the above-mentioned `X` but also to narrow down the filter and limit conditions in $Q2$. In the following, we denote by $result$ the set of intermediately extracted values of `a` that would be for sure included in the outcome of $Q1$. By $\langle lower, upper \rangle$, we denote ranges of the most-recently-executed rough query. In this case, the applied rough queries always refer to single numeric values, so parameters $lower$ and $upper$ should be interpreted as numeric values approximating the actual value from below and above, respectively. If `N` equals to 0 by the end of execution of Algorithm 1, then $result$ is the final answer to $Q1$ and no additional standard `select` statement is necessary. Otherwise, $result$ should be merged with the output of the following modification of $Q2$:

```
select distinct a from T
      where F and a > X and a < Y order by a desc limit N;
```

Algorithm 1 Rough pre-execution of $Q1$

Input: T, a, F, L; Output: X, Y, $result$, N

```
X ← +∞, Y ← +∞, B ← 1, M ← 0, N ← L, result ← ∅
while M < L do
  select roughly max(a) from T where F and a < X;
  X ← lower
  if B = 1 then
    if lower = upper then
      Y ← X, result ← result ∪ {X}, N ← N − 1
    else
      B ← 0
    end if
  end if
  select roughly count(distinct a) from T where F and a > X;
  M ← lower
end while
```

One may claim that the above mechanism should be rather implemented as one more internal method for improving the query execution. Actually, we intend to do it. However, there may be numerous other ways of using the knowledge grid that will need to wait for their implementation for a longer time. Rough SQL enables users to explore such methods by themselves, without waiting for the next Infobright's releases.

5 Further Extensions

5.1 Execution Internals

As already mentioned, rough queries work in their current form without accessing any packs of data. Precision of obtained $\langle lower, upper\rangle$ ranges depends on a *quality* of Infobright's knowledge grid, which intuitively corresponds to how *crisply* the packs' contents are described by their statistics (e.g. how close are the *min* and *max* values in the case of min/max statistics discussed in Section 3).

As emphasized in Section 4.2, we will continue developing algorithms producing better rough query outputs based on the knowledge grid. On the other hand, additional data access may not necessarily lead to a dramatic slowdown of a rough SQL execution. For instance, it is worth noting that Infobright caches recently used data packs in a memory. Integration of information residing within the knowledge grid with such packs may significantly improve a precision of results. Also, as one might notice in Section 3, Infobright's knowledge grid and information acquired from data accessed at a particular execution 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.

The above ideas can be applied in many ways. For instance, one may think about introducing a threshold for the overall execution time, which would yield a percentage of mostly useful data packs that can be analyzed at the exact level. We can also consider a design of Infobright-specific incremental querying (see Section 2). However, this would also require appropriate extensions of query syntax and user interfaces.

5.2 Semantics of Results

Interpretation of approximate results is quite natural for queries that return a single vector of values (e.g.: [`min(a)`, `sum(a)`] in the case of `select min(a), sum(a) from T where b > 1;`). The problems start for multi-tuple outputs (e.g. the query considered in Section 4.1). In the case of `group by` statements, if cardinality of the grouping columns is known to be small enough, one may report approximations for each of grouping values separately [5]. For queries with a known upper bound for the number of resulting tuples (e.g. queries with `limit`), one can still try to do the same but without a guarantee that the output approximations will describe the right tuples [13]. However, for complex queries with large amounts of output tuples there is a need to work simultaneously with approximations of all attributes defining the result.

Of course, more advanced users may find the currently implemented rough query results as not sufficiently informative. In some sense, we treat the whole output of a corresponding standard query counterpart as a single block of output tuples and we annotate it with very simple statistics. One of possible extensions is to consider a number of more fine-grained blocks and report statistics for each of them. One can compare it, to some extent, with the result grouping methods in Web search engines [14]. One can also define an optimization problem of clustering the output tuples in such a way that statistics of obtained blocks are maximally informative (see [16] where we considered an analogous task for loading data). However, it is crucial to remember that in the case of rough querying, statistics of output blocks need to be approximated well enough without (or almost without) explicit creation of the blocks themselves.

5.3 Inexact Querying

In the literature, approximate queries are usually supposed to terminate with tuples being almost the same as tuples resulting from the corresponding standard queries, sometimes with some additional confidence intervals. In [13], we followed this path by enriching Infobright's knowledge grid and the corresponding database engine functions in order to compute *degrees of (ir)relevance* of particular packs at different levels of a query execution. Then, basing on some analogies to extensions of rough sets and rough computing [8], 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 lower need for a data access.

In its current implementation, rough SQL is in some sense exact, i.e., there is 100% of certainty that actual results would drop into their approximations. However, depending on the future user requirements (compare with Section 4.2), we can extend our framework toward approximate queries resulting with crisper (e.g. by means of shorter $\langle lower, upper \rangle$ intervals) but not always fully accurate answers.

From this point of view, it is important to emphasize that all the above extensions can be introduced complementarily to each other. For instance, inexact $\langle lower, upper \rangle$ ranges can be used to describe output blocks introduced in Section 5.2. Also, our above-mentioned experiences with enriching the knowledge grid and new ways of its interaction with data can be combined with mechanisms proposed in Section 5.1.

6 Conclusions

We outlined the functionality and performance of rough SQL, which is available as an extension of standard SQL in Infobright's RDBMS. We discussed applications of rough queries in exploratory data analysis and processing, operational business intelligence, as well as in additional optimization of standard `select` statements.

We intend to continue collecting and categorizing practical use cases of rough SQL. We will also consider extending its current implementation according to the directions formulated in Section 5. Last but not least, we will attempt to integrate our rough query layer with some data mining and approximate reporting solutions [1,12].

References

1. 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)
2. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Data Mining and Knowledge Discovery 4(2/3), 89–125 (2000)
3. Naouali, S., Missaoui, R.: Flexible Query Answering in Data Cubes. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 221–232. Springer, Heidelberg (2005)
4. Zadrożny, S., Kacprzyk, J.: Issues in the Practical Use of the OWA Operators in Fuzzy Querying. Journal of Intelligent Information Systems 33(3), 307–325 (2009)
5. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation. In: SIGMOD, pp. 171–182 (1997)
6. Hu, Y., Sundara, S., Srinivasan, J.: Supporting Time-constrained SQL Queries in Oracle. In: VLDB, pp. 1207–1218 (2007)
7. Pawlak, Z.: Information Systems - Theoretical Foundations. Information Systems 6, 205–218 (1981)
8. Pawlak, Z., Skowron, A.: Rudiments of Rough Sets. Information Sciences 177(1), 3–27 (2007)
9. Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. Proceedings of VLDB Endowment 1(2), 1337–1345 (2008)
10. Ślęzak, D., Synak, P., Borkowski, J., Wróblewski, J., Toppin, G.: A Rough-columnar RDBMS Engine-A Case Study of Correlated Subqueries. IEEE Data Engineering Bulletin 35(1), 34–39 (2012)
11. Chaudhuri, S., Das, G., Narasayya, V.: Optimized Stratified Sampling for Approximate Query Processing. ACM Transactions on Database Systems 32(2), 9 (2007)
12. Cuzzocrea, A., Serafino, P.: LCS-Hist: Taming Massive High-dimensional Data Cube Compression. In: EDBT, pp. 768–779 (2009)
13. Ślęzak, D., Kowalski, M.: Towards Approximate SQL – Infobright's Approach. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS, vol. 6086, pp. 630–639. Springer, Heidelberg (2010)
14. Carpineto, C., Osiński, S., Romano, G., Weiss, D.: A Survey of Web Clustering Engines. ACM Computing Surveys 41, 1–38 (2009)
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. Ślęzak, D., Kowalski, M., Eastwood, V., Wróblewski, J.: Method and System for Database Organization. US Patent Application 2009/0106210 A1 (2009)