Philipp Rösch, Wolfgang Lehner

A Sample Advisor for Approximate Query Processing

Erstveröffentlichung in / First published in:

Advances in Databases and Information Systems: 14th East European Conference. Novi Sad, 20.-24.09.2010. Springer, S. 490-504. ISBN 978-3-642-15576-5.

DOI: http://dx.doi.org/10.1007/978-3-642-15576-5_37

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

A Sample Advisor for Approximate Query Processing

Philipp Rösch[1] and Wolfgang Lehner[2]

[1] SAP Research Center Dresden, Germany
philipp.roesch@sap.com
[2] Database Technology Group, Technische Universität Dresden, Germany
wolfgang.lehner@tu-dresden.de

Abstract. The rapid growth of current data warehouse systems makes random sampling a crucial component of modern data management systems. Although there is a large body of work on database sampling, the problem of automatic sample selection remained (almost) unaddressed. In this paper, we tackle the problem with a sample advisor. We propose a cost model to evaluate a sample for a given query. Based on this, our sample advisor determines the optimal set of samples for a given set of queries specified by an expert. We further propose an extension to utilize recorded workload information. In this case, the sample advisor takes the set of queries and a given memory bound into account for the computation of a sample advice. Additionally, we consider the merge of samples in case of overlapping sample advice and present both an exact and a heuristic solution. Within our evaluation, we analyze the properties of the cost model and compare the proposed algorithms. We further demonstrate the effectiveness and the efficiency of the heuristic solutions with a variety of experiments.

1 Introduction

Recent studies have revealed a rapid growth in current data warehouse databases regarding both the size and the number of queries. Additionally, more and more queries are of explorative nature where users browse through the data and search for interesting regions. [1]

In order to make the data exploration reasonably applicable, short response times are essential. However, the observed rapid growth conflicts with the need for short response times. A common solution for this problem is the use random samples. Initially, samples were mostly used for query optimization. In the last 10 years, however, the focus of database sampling has shifted more and more towards approximate query processing. Especially, precomputed and materialized samples provide large potentials due to their efficiency and their wide range of application. Several sampling schemes have been proposed that are optimized for different query types, like aggregation [2,3], group-by [4,5] or foreign-key joins [6,7]. While those sampling schemes provide great solutions for

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

single (groups of) queries, the more general problem of automatic sample selection for a specific mixture of queries or even an entire workload of a database is still an open issue.

In this paper, we address the problem of finding a set of samples for precomputation and materialization. We focus on simple random samples as those samples are easy to use and to maintain. Moreover, they can be used for a broad range of queries. Our solution is a sample advisor that suggests a set of samples for a set of queries specified by an expert (referred to as *expertise-based sample configuration*). The sample advisor is based on a novel cost model to evaluate a sample for a given query. This cost model allows us to give advice on a sample for an individual query. We further propose an extension to utilize recorded workload information (referred to as *workload-based sample configuration*). In this scenario, the sample advisor selects from all the pieces of sample advices those that minimize the runtime of the workload and fit into a given memory bound. A second strategy of this workload-based sample advisor additionally considers the merge of samples in case of overlapping sample advice.

In summary, we make the following contributions:

- We propose a cost model for the evaluation of a sample for a given query. With this cost model, we can give a piece of sample advice for an individual query (Section 2.1).
- Based on the cost model, we show how to compute an expertise-based sample configuration (Section 2.2).
- We further propose two strategies (with and without merging sample advice) to find a good workload-based sample configuration. For both strategies, we present an exact and a heuristic solution (Section 3).
- With a variety of experiments, we analyze the properties of the cost model and compare the proposed algorithms. We further demonstrate the effectiveness and the efficiency of the heuristic solutions (Section 4).

In Section 5, we discuss related work, and finally, we summarize the paper in Section 6.

2 Sample Advisor

Our major focus is the recommendation of an expertise-based sample configuration. We now define what a sample for an individual query should look like. In more detail, since we focus on simple random samples, we show how to find a proper sample size for a given query.

2.1 A Cost Model for Sample Selection

The sample selection problem is related to the physical design problem for indexes or materialized views. However, in the case of samples, we have to face up to a new dimension: As the result of a sample-based query is only an approximation, we additionally have to take a certain error into account. This error

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

can be an incompleteness—like for missing groups—or an inexactness—like for estimates of aggregates—and directly depends on the size of the sample.

Now, for the cost model, we have to discuss what makes up a good sample. Obviously, with a sample we want to achieve large decreases in the response times, and the memory cost should be low. These two goals ask for small sample sizes. At the same time, the estimates should be close to the actual values, and the results should be preferably complete. Clearly, these goals ask for large sample sizes. Hence, there is a conflict between the goals which has to be reflected by the cost model.

As a basis, we take the cost function of the DB2 Design Advisor [8] (bold part) and extend it by the approximation-related parts identified in this paper (italic part). The resulting cost model is:

$$\text{weight} = \frac{\textbf{decrease in response time} \cdot \textit{completeness}}{\textbf{memory cost} \cdot \textit{estimation error}}\,. \tag{1}$$

As can be seen, we append *completeness* (that we want to have) to the numerator and *estimation error* (that we don't like to have) to the denominator.

We now analyze the individual properties in more detail. Let N be the cardinality of the base data R, with $R = \{t_1, t_2, \ldots, t_N\}$. Further, let n be the cardinality of the sample S. Now, the sampling fraction f can be expressed as $f = n/N$. Moreover, let L denote the length of a tuple in the base data, while l is the length of a tuple in the sample.

Decrease in Response Time. For estimating the decrease in the response time, we make use of the simplified assumption that both the exact and the approximate query use table scans. Indeed, this assumption often holds in practice for the complex queries focused on in this paper. With this assumption the decrease in the response time Δt is proportional to $N-n$, and the relative decrease $\Delta t_{rel}(n)$ can be expressed as:

$$\Delta t_{rel}(n) = 1 - \frac{n}{N} = 1 - f\,. \tag{2}$$

Note that this function is independent from the dataset. It linearly decreases with increasing sample size.

Completeness. Let G be the set of groups defined by the query of interest. Then, $g_i \in G, i = 1...|G|$, denotes an individual group and $|g_i|$ denotes its size. Now, the probability p that at least one tuple of a group g_i is included into a sample of size n is given by:

$$p(g_i, n) = 1 - \frac{(N - |g_i|)!}{(N - |g_i| - n)!}\frac{(N-n)!}{N!}\,. \tag{3}$$

With this probability, the expected number of groups in the sample is

$$x(n) = \sum_{i=1}^{|G|} p(g_i, n)\,. \tag{4}$$

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

The completeness of an approximate query is the fraction of groups in the base data that are also in the sample:

$$c(n) = \frac{x(n)}{|G|}\,. \tag{5}$$

For selections, the completeness of the approximate result simply evaluates to f.

Memory Cost. The memory cost of a sample is made up of the number and the length of the tuples in the sample—as samples should be small, only required attributes are included into the sample. Hence, the absolute memory cost m_{abs} is given by $m_{abs}(n) = n \cdot l$. As for the decrease in response time, we can use the relative memory cost, which is given by:

$$m_{rel}(n) = \frac{n \cdot l}{N \cdot L}\,. \tag{6}$$

Estimation Error. Let $a_1, \ldots, a_l$ be the attributes that are aggregated in the query of interest. We now show how to compute estimation error for the AVG aggregate; the computation for the SUM aggregate is similar. For COUNT aggregates, the computation has to be adapted accordingly. We do not consider MIN or MAX as for these aggregation functions no estimation error can be computed.

Let RSD_j be the relative standard deviation of attribute a_j. Now, we can easily compute the relative standard error

$$RSE_{\hat{\mu}_j}(n) = RSD_j \sqrt{\frac{1}{n} - \frac{1}{N}} \tag{7}$$

which allows us to compare the error over multiple attributes. The overall estimation error over all the aggregation attributes is given by

$$RSE_{\hat{\mu}}(n) = \frac{1}{l} \sum_{j=1}^{l} RSE_{\hat{\mu}_j}(n)\,. \tag{8}$$

For queries with Group-By operations, the RSE is first computed for each group and then averaged over all groups. This can be done very efficiently in a single table scan by incrementally maintaining the size as well as the sum and the sum of squares of the aggregation attributes of each group [5].

Summing Up. We now put the pieces together. With the individual equations given above, the weight $w(n)$ of a sample of size n is computed by:

$$w(n) = \frac{\Delta t_{rel}(n) \cdot c(n)}{m_{rel}(n) \cdot RSE(n)}\,. \tag{9}$$

Note, this weight function is free of parameters (aside from the sample size) which was a main goal of our solution; we hold that parameters mainly confuse

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

the user. The weight of a sample represents its effectiveness. It reflects the quality of the result and the time saving as well as the price to pay.

This weight computation, however, has one shortcoming: The maxima of different samples differ both in amplitude and position, which makes comparisons of weights impractical. As a solution, we propose the following normalization:

$$\bar{w}(n) = \frac{w\,(a \cdot n)}{a} \tag{10}$$

with $a = \max(w(n))$. This normalization is based on the observation that high deviations of the data—and thus, large estimation errors—result in low weights. However, in order to provide good estimates, we need large samples for data with high deviations.

Based on the proposed weight function, we next introduce our sample advisor.

2.2 Expertise-Based Sample Configuration

In an expertise-based sample configuration, our sample advisor computes for each of the query given by the expert a piece of sample advice. Such a piece of sample advice comprises the base data R, the attributes A to be included into the sample and the sample size n, thus $SA = (R, A, n)$. The first two values can easily be derived from the query; the last one is determined with the weight of a sample given above. Hence, in order to compute the sample advice for a given query q, we first set R to the base data of q and A to the set of attributes referenced by q. Then, we determine the sample size with the following two steps:

1. Scan the base data of the query once and compute for each group the relative standard deviation and the size.
2. Iterate over different sample sizes and compute the weight. Remember the sample size with the largest weight.

The effort of the first step depends on the cardinality of the base data, and thus, is fixed. The effort of the second step, however, depends on the number of regarded sample sizes. As the weight function has a single maximum it can efficiently be found by algorithms like hill climbing or binary search.

The optimal expertise-based sample configuration SC_E now consist of all the samples specified by the pieces of sample advice of the expert-given queries.

3 Extension: Workload-Based Sample Configuration

Besides relying on an expert one common approach is to utilize recorded workload information. Clearly, in such a case we cannot materialize the samples for all the queries. We thus propose the following extension of our sample advisor. Note that this extension does not result in an optimal solution but shows how the approach proposed above can easily be adapted to workload-based scenarios.

Let the workload W be a (multi-)set of queries with $W = \{q_1, \ldots, q_k\}$. Before we compute the workload-based sample configuration SC_W, we preprocess

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

the workload by eliminating all but aggregation queries so that the workload only consists of queries relevant for approximate query processing. Further, the multiset of queries is transformed to a set by replacing the queries by (query, counter) pairs and merging duplicate queries. During this merge, predicates of the queries are ignored. [1] Hence, we get $W_{AQP} = \{(q_1, c_1), \ldots, (q_l, c_l)\}$.

With a memory constraint M, we get the following two steps:

1. Compute the optimal sample for each query of W_{AQP}. The result is a candidate set with pieces of sample advice $C = \{SA_1, \ldots, SA_k\}$.
2. Compute the optimal sample configuration SC_W of size M based on the candidate set C from the first step.

Obviously, with the first step we utilize our weight function and the computation of the expertise-based sample configuration. For the second step, we need a measure to compare different configurations. As a desirable sample configuration is characterized by a preferably low overall runtime of the workload, we use this overall runtime as our measure and try to minimize it.

As before, we assume table scans, and since we are not interested in actual response times, we simply use $r = n \cdot l$ as the response time for approximate queries and $r = N \cdot L$ for all queries with no sample in SC_W as they have to be answered with the base data. Now, the measure $\mathcal{F}$ of a sample configuration is:

$$\mathcal{F}(SC) = \sum_{q_i \in W_{AQP}} r_i \,. \tag{11}$$

The goal of the second step is to find the sample configuration that fits into M and minimizes $\mathcal{F}$. Obviously, the exact solution is an instance of the knapsack problem, which is known to be NP-hard.

Optimal Solution. To find the optimal solution (based on C), we consider all subsets C' of the candidate set C that fit into M and compute $\mathcal{F}$. The final sample configuration is the subset with the minimal measure $\mathcal{F}$.

Greedy Solution. The basic idea of the greedy approach is to successively add the most valuable candidates to the sample configuration until the memory bound is hit. We first order the candidate set C by the following score in descending order:

$$score(SA_i) = c_i \cdot \frac{N_i \cdot L_i}{n_i \cdot l_i} \,. \tag{12}$$

This score is composed of the query counter to account for the frequency of the sample usage—the more often a sample is used the more profitable gets its materialization—and the inverse of the relative memory cost. The second part of the score makes samples with smaller sampling fractions to be considered more valuable. This is motivated by the fact that those samples have low storage requirements and offer large response time benefits.

[1] We do not consider predicates as samples should be very general. Otherwise, samples would get something like materialized views.

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

Table 1. Sample advice candidates

Sample advice	Base data	Attributes	Sample size	Memory	Score
SA_1	R_1	$\{A_1, A_2\}$	4	8	62.5
SA_2	R_1	$\{A_3, A_4, A_5\}$	5	15	33.3
SA_3	R_1	$\{A_2, A_3, A_4\}$	7	21	23.8
SA_4	R_2	$\{A_1, A_3\}$	2	4	22.5

Next, we start with an empty sample configuration SC_W and successively add the candidates to SC_W in the given order until the next candidate does not fit into M. At this point, we allow to skip individual candidates in order to add those that still fit into M even if their score is lower.

Example 1. Consider 2 relations R_1 and R_2 with $N_1 = 100$ and $N_2 = 30$ as well as $L_1 = 5$ and $L_2 = 3$; for simplification, we say that the length of each attribute is 1 throughout the paper. With the candidates given in Table 1, let further $C = C_{ordered} = \{SA_1, SA_2, SA_3, SA_4\}$ be the (already ordered) candidate set. This table also shows the memory consumption—as defined by $n \cdot l$—and the scores of the candidates given that $c_i = 1$ for all $q_i \in W_{AQP}$ Now, let $M = 40$. We successively add the elements of $C_{ordered}$ to the initially empty SC_W. After having added SA_2, the memory consumption of SC_W is 23. Now, SA_3 does not fit into M while SA_4 does. Consequently, we skip SA_3 and add SA_4. The final sample configuration is $SC_W = \{SA_1, SA_2, SA_4\}$. The measure $\mathcal{F}$ of this sample configuration is $\mathcal{F}(SC) = 8 + 15 + 500 + 4 = 527$. □

3.1 Merging Pieces of Sample Advice

Besides the 'simple' selection of samples, we propose a second strategy that additionally considers the possibility of merging multiple pieces of sample advice. This idea is based on the following observation: In typical OLAP scenarios, many queries have the same base data—especially if the predicates are disregarded—and the referenced attributes often overlap. Hence, up to now there are samples in SC_W with the same base data and overlapping attribute sets. In order to effectively use the available memory, we consider to merge those samples. Then, the queries of both samples can be answered by the single merged sample; the redundancy in SC_W decreases.

Prerequisites for merging two pieces of sample advice SA_i and SA_j are the same base data $R_i = R_j$ as well as overlapping attributes: $A_i \cap A_j \neq \emptyset$. The merged piece of sample advice $SA_{i+j} = (R_{i+j}, A_{i+j}, n_{i+j})$ is computed by:

- $R_{i+j} = R_i = R_j$,
- $A_{i+j} = A_i \cup A_j$, and
- $n_{i+j} = \max\{n_i, n_j\}$.

When merging two pieces of sample advice, we take the maximum sample size for the following two reasons: First, decreasing n results in (considerably) higher

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

errors (estimation error and missing tuples) and second, increasing n has less impact on $\bar{w}$ than decreasing it.

However, aside from the prerequisites, one has to verify whether or not a merge is beneficial. Clearly, a query q_i of sample advice SA_i must read $A_{i+j} \setminus A_i$ additional attributes and $n_{i+j} - n_i$ additional tuples when using the sample of SA_{i+j} instead of the sample of SA_i. Having $\mathcal{F}$ in mind, a merge is only beneficial if the overall runtime of the workload decreases, and thus, if the merge frees enough memory to add an additional sample to SC_W.

Optimal Solution. For the optimal solution, we consider all possible merges. For each considered merge, we replace the respective pieces of sample advice by the merged piece of sample advice and proceed as in the strategy without merge. Obviously, this procedure is very expensive.

Greedy Solution. With the greedy approach here, we initially proceed as in the greedy approach without merge: We order the candidate set C by the *score* value and start by adding the pieces of sample advice into an initially empty sample configuration SC_W. However, when reaching a piece of sample advice that does not fit into the memory bound M, we now try to merge individual pieces of sample advice so that the current sample advice fits into M. If there is no merge for the current sample advice, we skip this piece of sample advice and proceed with the next one until all candidates are considered or the memory bound is hit.

Greedy Merge of Sample Advice. Recall that the goal of merging sample advice is to free enough memory to add the current piece of sample advice SA_i to the sample configuration SC_W. With the greedy approach, we proceed as follows: We consider all possible merges of the sample advice currently in $SC_W \cup SA_i$. From these merges, we choose the most beneficial one, i.e., the merge that frees the most memory. In the case of equal memory consumptions, we additionally consider the overall runtime $\mathcal{F}(CS)$. If the available memory is still too low, we again look for the most beneficial merge, but this time, we replace the two pieces of sample advice chosen to merge with the merged piece of sample advice. We repeat this procedure until either enough memory is freed or no more beneficial merges are possible. In the former case, we perform the merges, in the latter case, we skip the current piece of sample advice. In the greedy merge process, the repeated procedure of finding the most beneficial merge can be done very efficiently—all we need are the sample sizes, the tuple lengths and the overlap of the merge candidates.

Example 2. Consider again the setting of Example 1. We start by adding SA_1 and SA_2 to the initially empty sample configuration SC_W. Next, SA_3 is considered. SA_3 does not fit into M and thus, we try to find a merge. Since the attribute sets of SA_1 and SA_2 are disjoint, we do not merge these pieces of sample advice. SA_3, however, meets the condition to be merged with either SA_1 or SA_2 and we determine the more beneficial merge. For both SA_{1+3} and SA_{2+3}, we have $m_{abs} = 28$ and thus, we also have to consider the overall runtime. With

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

SA_{1+3}, the overall runtime of q_1, q_2, and q_3 sums up to $r = 28 + 15 + 28 = 71$, while with SA_{2+3}, the overall runtime is only $r = 8 + 28 + 28 = 64$. Thus, we prefer SA_{2+3} and we have $SC_W = \{SA_1, SA_{2+3}\}$ with a memory consumption of 36. In the next step, we add SA_4 to SC_W, and our final sample configuration is $SC_W = \{SA_1, SA_{2+3}, SA_4\}$. For this sample configuration, we get $\mathcal{F}(SC) = 8 + 28 + 28 + 4 = 68$, which is significantly lower—and thus, better—than $\mathcal{F}(SC) = 527$ of the greedy approach without merging sample advice. □

4 Experiments

We ran a variety of experiments to analyze the cost model and to compare the strategies for the construction of workload-based sample configurations.[2] We compared the strategy without merging samples (*NoMerge*) with the strategy that considers the merge of samples (*Merge*). We further evaluated the effectiveness and the efficiency of the heuristic algorithms (*NoMergeGreedy* and *MergeGreedy*). We experimented with well-defined synthetic datasets in order to discover the impact of certain "data formations" on the weight function and the resulting sample configuration. Finally, we ran experiments on a large real-world dataset consisting of retail data.

Note that the considered algorithms are deterministic with respect to the resulting sample configuration. Hence, our measure $\mathcal{F}$ for the comparison of the proposed algorithms can be computed analytically.

4.1 Experimental Setup

We implemented the sample advisor on top of DB2 using Java 1.6. The experiments were conducted on an Athlon AMD XP 3000+ system running Linux with 2GB of main memory.

Cost Model. For the evaluation of our cost model, we generated a small synthetic dataset R with $N = 1,000$ tuples and $L = 10$ attributes. The specific properties of this dataset are given in Table 2. Unless stated otherwise, the parameters take the value given in the last column (Default value).

Sample Configuration. For the evaluation of the workload-based sample configurations, we used two different datasets:

- A very small synthetic dataset with $N = 100$ tuples and $L = 15$ attributes. For this dataset, we used a workload of 5 carefully chosen queries.
- A large real-world dataset of retail data. The fact table of this dataset consists of $13,223,779$ tuples with $L = 15$ attributes (5 attributes are used for grouping and 8 for aggregation). Additionally, we chose 2 of the dimension tables which also consist of a few aggregation attributes. The workload of this dataset consists of 15 typical OLAP queries.

[2] We did not evaluate the expertise-based sample configuration as its computation is trivial once the cost model is given.

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

Table 2. Parameters for the experiments

Parameter	Range of values	Default value
Number of groups	$50 - 200$	100
Skew of group sizes	$0 - 1.4$	0.86
Average RSD	$5 - 50$	15

4.2 Analysis of the Cost Model

In the first part of our experiments, we analyzed the proposed cost model. We varied several parameters of the base data and computed the weight for samples of different sizes, each with $l = 4$ attributes.

Number of Groups. In the first experiment, we varied the number of groups from 50 to 200. Figure 1a shows the impact of the number of groups on the optimal sample size: For 50 groups, we get 5 tuples as optimal sample size, for 200 groups this values increases to 38. The reason is: more groups mean smaller groups which in turn are more likely to be missing in a sample. Thus, the optimal sample size increases with increasing number of groups.

Skew of Group Sizes. Next, we varied the skew of the group sizes. We chose a Zipfian distribution with z values ranging from $z = 0$ (uniform) to $z = 1.4$ (highly skewed). Here, the value of $z = 0.86$ results in a 90-10 distribution. The result is shown in Figure 1b. As can be seen, larger skews result in larger optimal sample sizes. Again, the reason is that smaller groups are more likely to be missing: The more skewed the group sizes, the more small groups in the base data. Hence, the larger the skew, the larger the optimal sample size.

Relative Standard Deviation of Aggregation Values. Finally, we varied the relative standard deviation of the base data. Inspired by our real-world dataset, we chose values from $RSD = 5$ to $RSD = 50$. As the RSD directly influences the estimation error (see (7)), larger RSDs result in larger estimation errors and thus, in larger optimal sample sizes. This is also shown in Figure 1c.

Summary. The results of the experiments show that the proposed cost model reflects the characteristics of the underlying data. Further, the advised sample sizes between about 0.5% and 5% look reasonable.

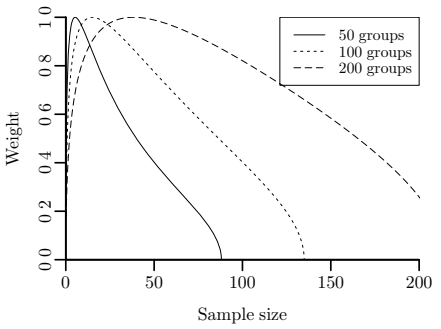
4.3 Sample Configuration

In the second part of our experiments, we compared the strategies and the algorithms for the computation of the sample configuration.

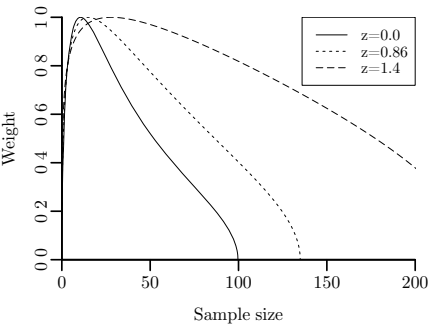
Synthetic Dataset. As stated above, we first evaluated our algorithms on a small dataset and we carefully selected 5 queries. With our cost model, we got the 5 pieces of sample advice illustrated in Figure 2 with the following scores:

Sample advice	SA_1	SA_2	SA_3	SA_4	SA_5
Score	2035.7	2000	1900	1000	1000

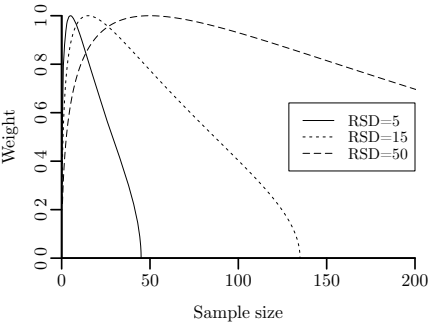
Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37



(a) Number of groups



(b) Skew of group sizes



(c) Rel. standard deviation

Fig. 1. Sample weight for various settings

Next, we computed the sample configuration with all four algorithms: *NoMerge*, *NoMergeGreedy*, *Merge*, and *MergeGreedy*. We varied the memory bound from $M = 0$ to $M = 79$ attribute values (for $M = 79$, all samples fit into the memory bound) and computed $\mathcal{F}$, i.e., the runtime of the 5 queries in terms of number of read attribute values. As can be seen in Figure 3a, there are some memory bounds where the merge of samples considerably decreases $\mathcal{F}$, e.g., for $M = 53$, the merge decreases $\mathcal{F}$ from $45,620$ to $18,552$ by a factor of about 2.5. The impact of the strategy and the greedy proceeding can be seen in the close-up on $\mathcal{F}$ for $M = 20$ to $M = 45$, see Figure 3b: For $M = 25$, the sample configurations of *NoMerge* and *NoMergeGreedy* consist of only SA_2 while *Merge* and *MergeGreedy* can significantly reduce $\mathcal{F}$ by merging SA_2 and SA_3. For $M = 28$, the sample configuration of *MergeGreedy* switches from SA_{2+3} to SA_1 and thus, it even performs worse than for $M = 27$. This is a drawback of the greedy proceeding. Furthermore, for $M = 30$ *NoMerge* selects SA_2 and SA_3 and thus, results in a better sample configuration than *MergeGreedy* that still selects SA_1. These results show that for our carefully chosen queries, the effectiveness of *MergeGreedy* (temporarily) may decrease for increasing memory bounds, and that *NoMerge* may be more effective than *MergeGreedy*. All in all, *Merge* always results in the best configuration, while *NoMergeGreedy* always results in the worst.

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

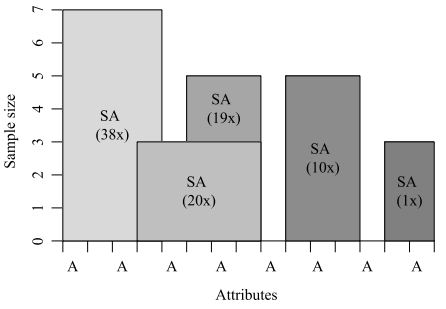
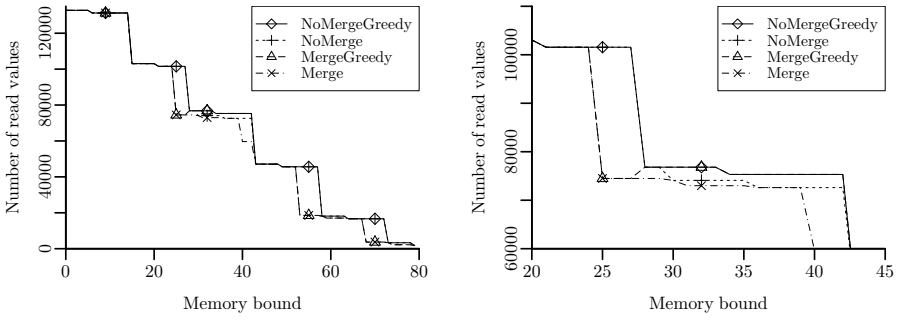


Fig. 2. Sample advice for the synthetic dataset



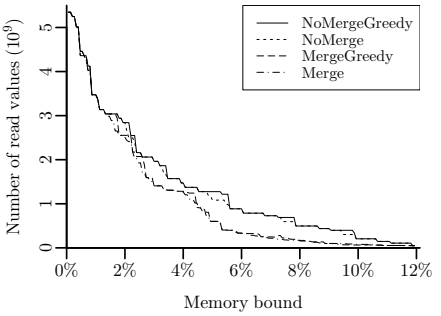
(a) Full range (b) Details

Fig. 3. Sample configurations for different memory bounds

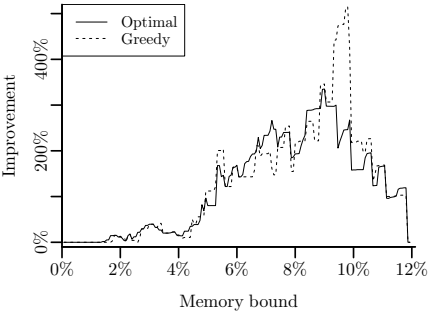
Real-World Dataset. Our next experiments were conducted on our large real-world dataset. The effectiveness of the different algorithms is given in Figures 4a and 4b. Note that in order to make the results easier to interpret, we used relative memory bounds in the plots. Again, we computed $\mathcal{F}$ for different memory bounds, see Figure 4a. Our results show that it is beneficial to merge sample advice. Further, they clearly demonstrate the effectiveness of the greedy algorithms. To make the benefit of merging sample advice even clearer, Figure 4b depicts the improvement achieved by merging sample advice. As can be seen, the improvement quickly reaches 100%, i.e., the number of attribute values that have to be read halves due to the merges. All in all, for a broad range of memory bounds it is very beneficial to consider merges of sample advice.

In a final experiment, we compared the efficiency of our algorithms. We varied the number of candidates from $|C| = 1$ to $|C| = 15$ and computed the sample configuration with all of our algorithms. Figure 5 illustrates the time to compute the sample configurations. These values are averaged over 50 runs, and we used a relative memory bound of 6%. The plot clearly shows the benefit of the greedy

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37



(a) Sample configurations for different memory bounds



(b) Improvement achieved by merging sample advice

Fig. 4. Real-real-world dataset

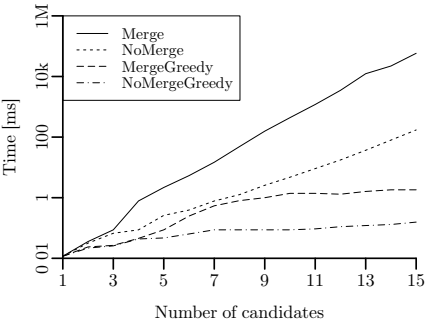


Fig. 5. Runtimes of the algorithms

proceeding: While the effort of the optimal solutions quickly gets high and exponentially increases with the number of candidates, the effort for the greedy solutions is significantly lower. For *NoMergeGreedy*, the effort is logarithmic due to the ordering of the candidates, and for *MergeGreedy*, the effort is quadratic due to the greedy merge. In an additional run, we measured the times for the greedy algorithms for $|C| = 75$ candidates. Here, the computation of the sample configuration still took less than a millisecond for *NoMergeGreedy* and about 1.3 seconds for *MergeGreedy*.

Summary. Our results on synthetic and real-world datasets show that the merge of sample advice is beneficial. It significantly reduces the runtime of the given workload. Additionally, the greedy algorithms are very efficient and effective—they often result in the same sample configurations as the exhaustive approaches while requiring only a fraction of the time to compute the configuration.

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

5 Related Work

In this section, we briefly review related work in the field of automatic sample selection for approximate query processing in databases which is barely studied. Some initial ideas in this field are those of [9] and [10]. The strategy of the technique shown in [9] is to select from all possible synopses those that influence the query plan or the execution costs. However, the problem of a memory bound and hence, the partition of the available space is not regarded. The solution in [10] proposes a technique for the selection of synopses as well as for the partitioning of the available memory. However, all the considerations build on spline-based synopses, so that the solutions cannot easily be translated into samples. Moreover, the focus of both solutions is the selectivity estimation where the approximation error bears another meaning as it is not directly passed to the user.

The problem is also related to the physical design problem for indexes [11] and materialized views [12]. Most of these solutions use a what-if interface [13] and ask the optimizer for the benefit. However, optimizer calls are expensive and estimated costs may be far off [14]. Moreover, the extension of the what-if interface in order to estimate both the cost and the error introduced by approximate query processing might be a complex task. The alternative solution is to define an explicit cost model as done by the approaches in [12,8]. Our weight function was inspired by that of [8] but had to be extended for both the estimation error and the incompleteness.

6 Summary

In this paper, we proposed a sample advisor for the approximate answering of analytical queries. This sample advisor is based on a novel cost model for the sample selection. We proposed a weight function that enables us to give sample advice for any individual query. Building on that, we regarded two different scenarios: (i) an expertise-based sample configuration for individually specified queries, and (ii) a workload-based sample configuration for a recorded workload regarding a given memory bound. For the computation of the workload-based sample configuration, we presented two strategies: The first strategy selects from the available sample advice those pieces of advice that minimize the runtime of the workload. The second strategy provides a more sophisticated solution by merging pieces of sample advice, which may significantly reduce the overall runtime of the given workload. For both strategies, we presented and evaluated an exact and a heuristic algorithm. Our experiments have shown that the merge of samples is almost always beneficial and provides large runtime savings for the given workload. Furthermore, our greedy algorithms significantly reduce the computation cost with only low impact on the effectiveness.

Acknowledgment. We thank Torsten Weber for his inspiring ideas.

Final edited form was published in "Advances in Databases and Information Systems: 14th East European Conference. Novi Sad 2010", S. 490-504, ISBN 978-3-642-15576-5
http://dx.doi.org/10.1007/978-3-642-15576-5_37

References

1. Winter, R.: Scaling the Data Warehouse. Intelligent Enterprise (2008), http://www.intelligententerprise.com/showArticle.jhtml?articleID=211100262
2. Chaudhuri, S., Das, G., Datar, M., Narasayya, R.M.V.: Overcoming Limitations of Sampling for Aggregation Queries. In: ICDE, pp. 534–544 (2001)
3. Rösch, P., Gemulla, R., Lehner, W.: Designing Random Sample Synopses with Outliers. In: ICDE, pp. 1400–1402 (2008)
4. Acharya, S., Gibbons, P., Poosala, V.: Congressional Samples for Approximate Answering of Group-By Queries. In: SIGMOD, pp. 487–498 (2000)
5. Rösch, P., Lehner, W.: Sample Synopses for Approximate Answering of Group-By Queries. In: EDBT, pp. 403–414 (2009)
6. Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: Join Synopses for Approximate Query Answering. In: SIGMOD, pp. 275–286 (1999)
7. Gemulla, R., Rösch, P., Lehner, W.: Linked Bernoulli Synopses: Sampling Along Foreign-Keys. In: SSDBM, pp. 6–23 (2008)
8. Zilio, D.C., Zuzarte, C., Lohman, G.M., Pirahesh, H., Gryz, J., Alton, E., Liang, D., Valentin, G.: Recommending Materialized Views and Indexes with IBM DB2 Design Advisor. In: ICAC, pp. 180–188 (2004)
9. Chaudhuri, S., Narasayya, V.: Automating Statistics Management for Query Optimizers. IEEE Trans. on Knowl. and Data Eng. 13(1), 7–20 (2001)
10. König, A.C., Weikum, G.: A Framework for the Physical Design Problem for Data Synopses. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 627–645. Springer, Heidelberg (2002)
11. Chaudhuri, S., Narasayya, V.R.: An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In: VLDB, pp. 146–155 (1997)
12. Gupta, H., Mumick, I.S.: Selection of Views to Materialize in a Data Warehouse. IEEE Trans. on Knowl. and Data Eng. 17(1), 24–43 (2005)
13. Chaudhuri, S., Motwani, R., Narasayya, V.: Random Sampling for Histogram Construction: How much is enough? In: SIGMOD, pp. 436–447 (1998)
14. Gebaly, K.E., Aboulnaga, A.: Robustness in Automatic Physical Database Design. In: EDBT, pp. 145–156 (2008)