Rainer Gemulla, Henrike Berthold, Wolfgang Lehner

Hierarchical Group-Based Sampling

Erstveröffentlichung in / First published in:

Database: Enterprise, Skills and Innovation.22nd British National Conference on Databases. Sunderland, 05. - 07.07.2005. Springer, S. 120-132. ISBN 978-3-540-31677-0.

DOI: https://doi.org/10.1007/11511854_10

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

Hierarchical Group-Based Sampling

Rainer Gemulla, Henrike Berthold, and Wolfgang Lehner

Dresden University of Technology
Database Technology Group
{gemulla,henrike.berthold,lehner}@inf.tu-dresden.de

Abstract. Approximate query processing is an adequate technique to reduce response times and system load in cases where approximate results suffice. In database literature, sampling has been proposed to evaluate queries approximately by using only a subset of the original data. Unfortunately, most of these methods consider either only certain problems arising due to the use of samples in databases (e.g. data skew) or only join operations involving multiple relations. We describe how well-known sampling techniques dealing with group-by operations can be combined with foreign-key joins such that the join is computed after the generation of the sample. In detail, we show how senate sampling and small group sampling can be combined efficiently with the idea of join synopses. Additionally, we introduce different algorithms which maintain the sample if the underlying data changes. Finally, we prove the superiority of our method to the naive approach in an extensive set of experiments.

1 Introduction

As a result of rising computation and storage capacities, data acquisition has become simpler and more versatile. The amount of information stored on a wide range of different media has increased tremendously during the past years [1]. Data warehouse systems integrating different databases are capable of persistently storing this surge of information. However, it is rather difficult to extract knowledge from these voluminous databases, since the respective database queries usually suffer from long runtimes. Often an approximate but fast answer is the better alternative, e.g. to support interactivity. Sampling is a widely used technique which balances query result accuracy and response time.

The well-known simple random sampling (SRS) selects a fixed-sized random subset of a relation such that every possible subset has the same probability of being drawn. Approximate query evaluation using SRS assumes that the underlying data is uniformly distributed. In order to circumvent this restriction and to extend SRS to multiple relations, several techniques have been proposed. However, they only address either data distribution or join processing. We show how to combine sampling techniques developed to accurately answer group-by queries [2, 3] with the well known technique of join synopses [4] for foreign-key joins.

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

Related Work. The New Jersey Data Reduction Report [5] provides an overview of approximate query processing in general. Database-specific sampling techniques can be divided into two classes: *online sampling*, which computes the sample at query execution time, and *offline sampling*, which pre-computes the sample and materializes it in the database. Obviously, by using offline sampling it is possible to spend more effort in the computation of the sample in order to increase the accuracy of approximate results. However, the sample has to be maintained if the data is modified.

The method of *online aggregation* [6] belongs to the former of the two classes mentioned above. The main idea is to present the user with iteratively refined approximate results for aggregation queries. However, most sampling techniques generate the sample offline in order to deal with data skew (e.g. non-uniformly distributed value frequencies). *Reservoir sampling* [7, 8] allows the computation of a sample of predefined size. *ICICLES* [9], developed by Ganti et. al., attempts to generate and maintain a sample tailored to the actual query workload. The *outlier indexing* [10] detects outliers within the data and uses this knowledge for sample computation.

The main problem of bringing together sampling and join is the fact that these two operations do not commute. For two relations R_1 and R_2, it holds in general:

$$SRS(R_1 \bowtie R_2) \neq SRS(R_1) \bowtie SRS(R_2)$$

Therefore, it is not possible to compute a sample of a join by only using the samples of the participating relations [11]. Fortunately, in the common case of an N:1-relationship as appearing in star and snowflake schemes the situation is less difficult, because it is possible to sample at least one of the involved relations:

$$SRS(R_1 \bowtie_{N:1} R_2) = SRS(R_1) \bowtie_{N:1} R_2$$

This property is the foundation of *join synopses* [4] which pre-calculate samples over foreign-key relationships. With the help of this technique, expensive joins are avoided at query execution time (sec. 2.1).

Typically, data is not uniformly distributed. This data skew causes enormous problems if sampling is not applied carefully. For instance, the *small group problem* appears: if the values of the grouping attributes are not uniformly distributed with regard to their frequency, groups consisting of only a few tuples appear infrequently in the sample and, thus, contribute to the approximate result infrequently. Group-based sampling techniques such as *senate sampling* [2] and *small group sampling* [3] deal with this problem.

Outline. The remainder of the paper is organized as follows. Section 2 provides an overview of fundamental techniques which deal with sampling, join and group-by. Additionally, we introduce the concept of a foreign-key tree which orders relations into a hierarchy (thus the name), and we explain a naive combination approach. In sections 3 and 4, we introduce algorithms superior to the naive one. Section 5 presents an extensive experimental evaluation. Finally, a summary concludes the paper in section 6.

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

2 Sampling, Join and Group-by

This section briefly introduces sampling techniques for join operations as well as group-based sampling. The new concept of *foreign-key trees* serves as the foundation for the combination of these techniques.

2.1 Join Synopses

Acharya et. al. combine sampling and foreign-key joins by creating so-called *join synopses* [4]. The foreign-key relationship of a relation R_1 with foreign key fk to a relation R_2 is denoted $R_1 \rightarrow_{fk} R_2$. The symbol $\Rightarrow$ denotes the transitive closure of $\rightarrow$, and $\Rightarrow^*$ the reflexive transitive closure. A foreign-key graph visualizes $\rightarrow$ over a schema (cf. Fig. 1, left). Every node represents a relation (and vice versa), every edge models a foreign-key relationship. The example shows a relation A which has two foreign keys fk_1 and fk_2 to relation B, i.e., A references B twice.

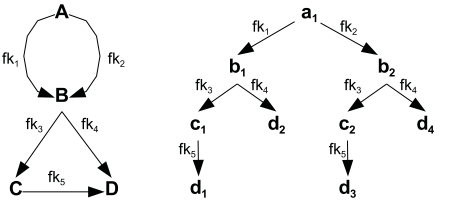

Fig. 1. F.k. graph (left) & tree (right)

A join $R_1 \bowtie R_2$ is a *foreign-key join* (FKJ) with *source relation* R_1, if the join condition compares a foreign key of R_1 with the primary key of R_2 ($R_1 \rightarrow R_2$) for equality. The result of the FKJ consists of the primary key of the source relation, and the foreign keys of all involved relations. If it is joined with additional relations by using one of its foreign keys, another FKJ with the same source relation is created – thus, there is always exactly one source relation, which is used as a starting point. Between a relation R and an FKJ with source relation R, there is a 1:1-relationship. Informally, the FKJ looks up foreign keys in the respective relations and extends each tuple of the source relation by the result. A simple example scenario is shown in Figure 2. On the right, the result of the FKJ $Emp \bowtie Dep \bowtie Loc$ is presented.

In the following, we assume that the foreign-key graph is free of cycles. In this case, a *maximum foreign-key join* (MFKJ) can be determined for every relation. It is free of redundancy; for example, it eliminates joins like $A \bowtie_{fk_1} B \bowtie_{fk_1} \ldots \bowtie_{fk_1} B$ (cf. Fig. 1, left). We introduce *foreign-key trees* to model such maximum foreign-key joins.

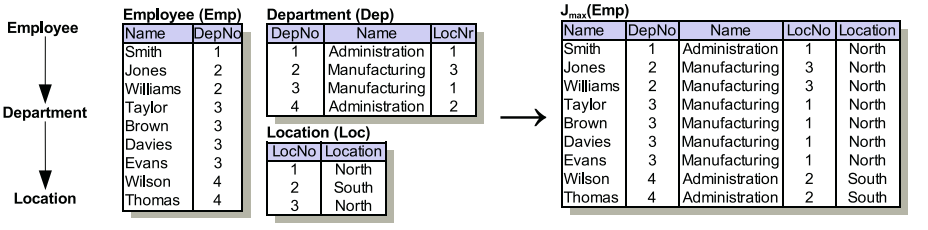

Fig. 2. Example scenario and maximum foreign-key join

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

Definition 1. *The* foreign-key tree *tree*(R) *of a relation* R *is made up of a root node* r *with associated relation* $rel(r) = R$*. For each foreign-key relationship of type* $R \rightarrow_{fk} S$*, the foreign-key tree of* S *is connected to* r *with the edge* $r \rightarrow_{fk}$ *tree*(S)*.*

Figure 1 (right) shows the foreign-key tree (FKT) of relation A. There is an N:1-relationship between nodes and relations. The MFKJ consists of one join operation per edge of the foreign-key tree. In detail, each node is joined with the MFKJ of its successors[1]. The MFKJ $J_{max}(A)$ of relation A is given by:

$$J_{max}(A) = A \bowtie_{fk_1} (B \bowtie_{fk_3} (C \bowtie_{fk_5} D) \bowtie_{fk_4} D) \\ \bowtie_{fk_2} (B \bowtie_{fk_3} (C \bowtie_{fk_5} D) \bowtie_{fk_4} D)$$

The *join synopsis* of a relation R resembles a random sample of $J_{max}(R)$. Due to the 1:1-relationship of tuples in R and $J_{max}(R)$, it is possible to sample R before executing the join, i.e., the sample $U = SRS(R)$ is computed and afterwards, it joined with all the relations of the FKT, yielding $J_{max}(U)$. This is crucial for the practicability of the method, since the complete MFKJ is very expensive to obtain. If a join synopsis is created for every relation of a schema, it is possible to approximately answer all queries with FKJs by using the appropriate synopsis.

2.2 Senate Sampling

Senate sampling [2] attacks the problem of sampling small groups; it ensures that all groups appear in the sample. This is achieved by assigning the same amount of space in the sample to each group. Therefore, it is necessary that all potential grouping attributes are already known in advance. Furthermore, the number of the non-empty groups has to be smaller than the size of the sample. Thus, it is guaranteed that for every group there is at least one tuple reserved in the sample.

The sampling process requires one scan of the relation. For each group, the algorithm creates an independent reservoir [7], i.e. a temporary relation usually stored in main memory. At any time, the reservoir contains a random sample of all tuples of its group. If m groups have been seen so far, the size of each reservoir is limited to $s_g = \frac{n}{m}$ tuples with n being the sample size. Therefore, each first occurrence of a group yields to a decrease of the size of all reservoirs. After all tuples have been processed, the reservoirs are written into a single sample table.

By using the senate sample, group-by queries can be answered approximately and without losing any group. However, the procedure has some inherent difficulties: it is often the case that the number of groups is too high because of the consideration of all potential grouping attributes and, thus, no useful sample can be generated. Alternatively, it is possible to compute multiple samples, each

[1] In the following, the term "node" is used synonymously for the relation associated with it; e.g., the tuples of a node a refers to the tuples of the relation A assigned to that node, actually

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

for a subset of the grouping attributes. Furthermore – if we limit the number of grouping attributes – small groups contain usually far less tuples than they have space in the sample. As a result, parts of the sample remain unused.

2.3 Small Group Sampling

Another approach to solve the small-group problem has been developed by Babcock et.al. and is called *small group sampling* (SGS, [3]). It generates multiple sample tables and selects an adequate subset of them at query evaluation time. The basic idea of SGS is to create a so-called *small group table* (SGT) for each attribute of the base relation. These small group tables include all tuples which have a rare value in the respective attribute. Therefore, the SGTs of a query's grouping attributes consist of tuples belonging to small groups. Additionally, a random sample of the base relation is generated. In general, this *base sample* covers tuples which belong to large groups.

The user has to define three parameters for the generation of the small group sample. First, the *base sampling rate* r $(0 < r \leq 1)$ determines the size n of the base sample in dependency of the number N of the tuples in the base relation $(n = N \cdot r)$. Second, if an attribute has more than τ distinct values, no SGT is generated. Therefore, τ is used to determine which attributes are likely to appear in a group-by clause. Finally, the *small group fraction* f defines the upper size limit of an SGT $(n_{SGT} \leq N \cdot f)$. Only the most rare attribute values appear in the SGT. Thus, f implicitly draws the line between rare and frequent.

The computation of the SGS consists of two phases, each requiring one table scan. First, a histogram is generated for each attribute. With their help, it is possible to decide which values are rare. In the second phase, this knowledge is used to generate the base sample and the SGTs. At query processing time, the base sample and the SGTs of the respective grouping attributes are used for approximate query evaluation. All tuples of groups which consist of at least one rare value in a grouping attribute are completely covered by an SGT – their aggregate is calculated exactly. All other groups are served by the base sample and evaluated approximately.

Dependencies between attributes may cause the SGS to miss some groups. Often, small groups which consist of frequent values only are not represented in the sample. But in contrast to senate sampling, the grouping attributes do not have to be known in advance and, thus, a small group sample is designed for arbitrary grouping attributes. However, the parametration is quite difficult. Additionally, functional dependencies between attributes lead to redundant SGTs, e.g., the SGT of an attribute *country_name* is likely to be equal to that of the attribute *country_code*.

2.4 Naive Combination

By combining join synopses and group-based sampling, we are able to answer queries with foreign-key joins and/or group-by approximately. It is not meaningful to simply use the MFKJ of a senate or small group sample, since thereby

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

only attributes of the source relation would be considered as potential grouping attributes. Alternatively, the complete MFKJ of the source relation could be calculated and sampled afterwards. This *naive approach* is not feasible in most cases due to its high computation costs. Instead, we introduce new algorithms called *hierarchical senate sampling* (HSEN) and *hierarchical small group sampling* (HSGS), which result in samples identical to that of the naive approach but which are more efficient to obtain. Therefore, we show how to pull sampling before the join as done for join synopsis computation.

3 Hierarchical Senate Sampling

HSEN requires the FKT of the source relation as its input. Unlike in regular senate sampling, grouping attributes are defined at node level. Therefore, it is possible to assign different grouping attributes to relations that appear more than once within the tree. Throughout the paper, we use the scenario shown in Figure 2 as an example. Note that the foreign-key graph and the foreign-key tree of *Emp* are equal. Futhermore, let *Loc.Location* and *Dep.Name* be grouping attributes. In the following, we describe the computation of HSEN, which is divided into two phases.

3.1 Phase 1: Group Tables

Regular senate sampling determines the group of each tuple by extracting the values of the grouping attributes. Unfortunately, this is not possible if multiple relations are involved. Therefore, additional information has to be gathered from the data before sampling the source relation.

Definition 2. *The* group table GT_u *of a node* u *contains one entry for every tuple of* u*. Each of these entries consists of a primary key and a group identifier (GID).*

Thus, the group table (GT) captures the relationship between tuples and groups (cf. Fig. 3, left), which in turn are represented by a unique group identifier (GID). With their help, the group of a tuple of the source relation can be determined by looking up its foreign keys in the GTs of the respective referenced nodes. The actual values of the grouping attributes are not of interest. It is sufficient that tuples belonging to the same group have the same GID, and that tuples belonging to different groups have a different GID as well.

The group table does not have to be calculated for every node. A node is called *directly grouping-relevant* if at least one grouping attribute has been defined on it. In the example, this applies to the nodes *Loc* and *Dep*. Additionally, a node is called *indirectly grouping-relevant* if one of its successor nodes is grouping-relevant. This holds for *Dep* and *Emp*.

The first phase of HSEN computes the GTs of all grouping-relevant, direct successor nodes of the source relation. The algorithm starts with those nodes

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

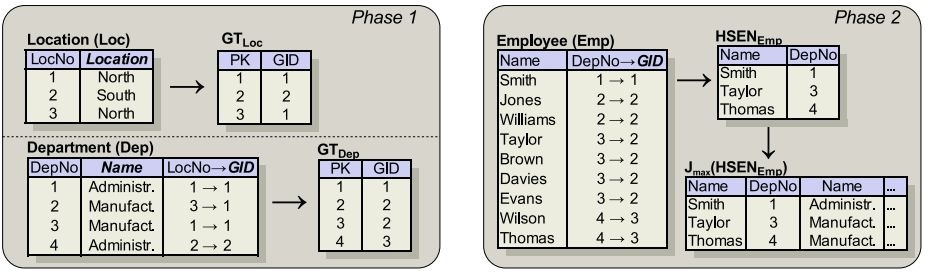

Fig. 3. Computation of the hierarchical senate sample

which do not have any grouping-relevant successor, and subsequently, proceeds backwards along the foreign-key relationships to the root node (bottom-up).

Let u be the node currently processed and let $v_1, \ldots, v_k$ be its direct, grouping-relevant successors. The computation of GT_u requires one scan of u. If $k = 0$, i.e. u has no successor nodes with grouping attributes, each tuple's group is determined by simply using the values of the grouping attributes defined on it. A unique GID is assigned to each group. Thereby, a temporary data structure captures the 1:1-relationship between groups and GIDs. For each tuple, an entry consisting of its primary key and its GID is included in the GT. For instance, Fig. 3 shows the GT of *Loc*. The relationship between groups and GIDs is $\{(North, 1), (South, 2)\}$.

In order to calculate the GT of an indirectly grouping-relevant node ($k > 0$), the GTs of all its direct, grouping-relevant successors must be known. For example, GT_{Loc} is required to calculate GT_{Dep}. In general, the procedure is identical to the one for $k = 0$. However, the value of every foreign key to a successor node is looked up in the respective group table. Afterwards, the obtained GID is used as an *additional* grouping attribute. Therefore, groups generated by successor nodes are considered, too. The complete algorithm is presented in more detail in the full paper [12].

The GT of the node *Dep* is shown in the lower left of Fig. 3. For its computation, the attribute *Name* and the GID out of GT_{Loc} have been used as grouping attributes. By now, GT_{Loc} is not needed anymore and is therefore deleted. The first phase is finished at this point, since there is no need to calculate a GT for the root node.

3.2 Phase 2: Sampling

The sampling of the source relation is almost identical to regular senate sampling. The only difference is that foreign keys have to be looked up in the GT of the respective successor node, and the obtained GID has to be used as additional grouping attribute. Due to space restrictions, this algorithm is not presented here.

Figure 3 (right) depicts a possible sample of size $n = 3$. One tuple is sampled from each of the groups (Adm, N), (Man, N), and (Adm, S). Finally, the MFKJ of this sample has to be calculated. In the full paper [12], we discuss several

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

optimizations of the algorithm presented here and describe how to maintain the sample incrementally.

4 Hierarchical Small Group Sampling

As with senate sampling, SGS can be naively extended to multiple relations using the MFKJ of the source relation. By proceeding hierarchically, the computation costs are lowered noticeably and, thus, the method becomes practicable. Two core problems have to be solved: the determination of rare values on the one hand and of all tuples with these values on the other hand. The hierarchical approach is structured into 3 phases: phase 1 deals with the former of the two problems, phase 2 with the latter. In Phase 3 the base sample and the SGTs are computed.

4.1 Phase 1: Histogram Calculation

As explained in section 2.3, the determination of rare values requires a histogram for each attribute. If there is only one relation R, their calculation is simple. However, if multiple relations have to be considered, it is not possible to calculate the histograms[2] of every relation separately, since the influence of the foreign keys is lost otherwise. Instead, the number of references to every tuple has to be considered during histogram calculation.

Definition 3. *The* reference table $RT_{u \Rightarrow v}$ *contains the primary key of every tuple of node* v *together with the number of tuples of node* u *that reference it via foreign keys* $(u \Rightarrow v)$*. Non-referenced tuples do not appear in the reference table.*

For example, Fig. 4 (upper left part) shows the reference table $RT_{Emp \Rightarrow Dep}$, which captures the number of references of every tuple in Dep. For instance, the department with the number 3 is referenced four times. Subsequently, the reference table is used for histogram calculation. Let w be the root node of the FKT. Then, the reference table $RT_{w \Rightarrow v}$ is required for the computation of the histograms of a node v. The reference count is used as a weight for each tuple. For instance, the tuple $(3, Adm, 1)$ of Dep is counted four times.

The computation of histograms and reference tables is done simultaneously. In fact, the RT of a node v equals the (weighted) histogram of the foreign key attributes of its predecessor x, that is, the reference table $RT_{w \Rightarrow v}$ is computed together with the histograms of x. Since the root node w has no predecessor, its histograms and reference tables to successor nodes are computed first and without any weighting. Subsequently, the FKT is traversed top-down. For each tuple of a node v, its primary key is looked up in the reference table $RT_{w \Rightarrow v}$, and the obtained reference count is used as tuple weight, i.e. as scale factor. Therefore, the histograms are identical with those of the respective attributes in the MFKJ $J_{max}(rel(w))$. Please refer to the full paper [12] for a more detailed description of this algorithm.

[2] The histogram of the attribute i of node u is denoted $H_{u,i}$

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

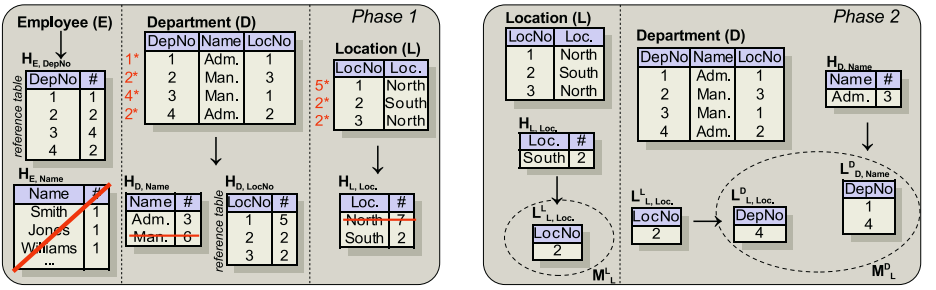

Fig. 4. Phase 1 and 2 of the computation of the hierarchical small group sample

After processing a node u, $RT_{w \Rightarrow u}$ is not needed anymore and therefore deleted. Furthermore, all histograms are restricted to rare values, i.e. iteratively, the most frequent value is removed until the number of tuples represented by the histogram does not exceed the upper size limit anymore. All remaining tuples will be included in the SGT later on (cf. sec. 2.3). Attributes with more than τ distinct values or without any rare values do not require a histogram anymore.

In the left of Figure 4, the first phase of hierarchical small group sampling is illustrated. In the example, the maximum number of distinct values is set to 2, the upper size limit of an SGT to 3. Therefore, only the histograms of the attributes *Dep.Name* and *Loc.Location* remain.

4.2 Phase 2: Key Sets and Assignment Sets

An SGT has to be generated for each attribute for which there is a non-empty histogram remaining after the first phase. Now, we have to determine which tuples of the source relation have to be included in which SGTs. Thus, the restricted histograms are converted step by step to so-called *key sets* (KS, per attribute), which consist of the primary keys of all tuples with rare values.

Definition 4. *The* key set $L^u_{u,i}$ *of a histogram* $H_{u,i}$ *contains the primary keys of all tuples of* u*, whose value of the attribute* i *appears in the histogram.*

Since the restricted histograms contain rare values only, this applies to the key sets, too. All histograms but those of the root node have to be converted to key sets according to definition 4. Thereby, the FKT is traversed bottom-up and a second table scan is performed at every node with at least one non-empty histogram. Figure 4 (right) shows the key sets $L^{Loc}_{Loc,Loc.}$ and $L^{Dep}_{Dep,Name}$ for the example scenario. According to these key sets, the location with primary key 2 as well as the departments with primary keys 1 or 4 have a rare value in the attribute *Location* and *Name* respectively (cf. Fig. 2).

Additionally, when processing a node v the key sets of its direct successor nodes have to be converted to the primary keys of v.

Definition 5. *Let* v*,* u*, and* x *be three nodes in the foreign-key tree with* $v \neq u$ *and* $v \rightarrow u \Rightarrow^* x$*, and assume that* $L^u_{x,i}$ *is known. Then, the key set* $L^v_{x,i}$ *contains*

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

the primary keys of all tuples from v, whose foreign keys to u appear in the key set $L^u_{x,i}$.

To summarize, the key set $L^v_{x,i}$ contains all those primary keys of v which lead to rare values of the attribute i of node x. The key sets of the leaf nodes (covered by definition 4) are used as a starting point. In this case, v and x refer to the same node. For instance, it holds $Dep \rightarrow Loc \Rightarrow^* Loc$ in the example scenario. Therefore, the key set $L^{Loc}_{Loc,Loc.}$ is converted to $L^{Dep}_{Loc,Loc.}$ according to definition 5. It contains the primary keys of all tuples of Dep whose foreign key to Loc is present in the key set $L^{Dep}_{Loc,Loc.}$. In other words, the department with the primary key 4 leads to a rare value in the $Loc.Location$ attribute.

An *assignment set* (AS, per node) integrates all the key sets of a specific node.

Definition 6. *The* assignment set M^u_L *of a node u consists of the key sets $L^u_{u,i}$ of all restricted histograms $H_{u,i}$. If u has the direct successor nodes $v_1, \ldots, v_k$, M^u_L additionally contains all key sets out of $M^{v_1}_L, \ldots, M^{v_k}_L$ converted to primary keys of u according to definition 5.*

Thus, all key sets of the assignment set M^u_L contain primary keys of u only. In the example scenario, there are two AS: $M^{Loc}_L = \{L^{Loc}_{Loc,Loc.}\}$ and $M^{Dep}_L = \{L^{Dep}_{Dep,Name}, L^{Dep}_{Loc,Loc.}\}$ (cf. Fig. 4). The computation of an AS requires the AS of all successor nodes. This is the reason why the FKT has to be processed bottom-up. Note that after the assignment set of a node u has been created, neither its histograms nor the assignment sets of its successors are needed anymore. Please refer to the full paper [12] for further details.

Only the assignment sets of the direct successors of the root node are the output of the second phase. Subsequently, they are used to sample the source relation.

4.3 Phase 3: Sampling

Just as with regular SGS, the source relation is scanned once to compute the base sample and the SGTs. For each attribute of the source relation, the assignment of tuples to SGTs is done with the help of its histogram. For all other attributes, the assignment sets are used, that is, for each direct successor node v the respective foreign key of the current tuple is extracted and looked up in every key set $L^v_{x,i}$ within M^v_L. If it is present in there, the current tuple is copied to the SGT of attribute i of node x. For example, only the tuples with department number 1 or 4 are included in the SGT of $Dep.Name$ (Fig. 5) since $L^{Dep}_{Dep,Name}$ contains the keys 1 and 4 only.

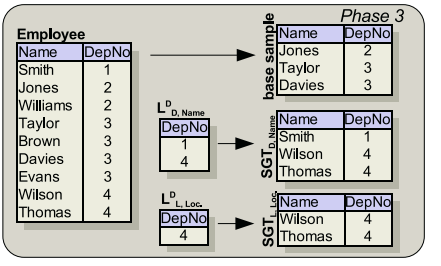

Fig. 5. Phase 3 of HSGS

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

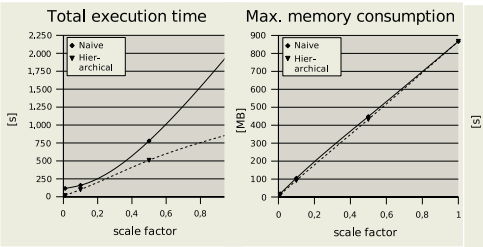

Fig. 6. DB size & computation costs

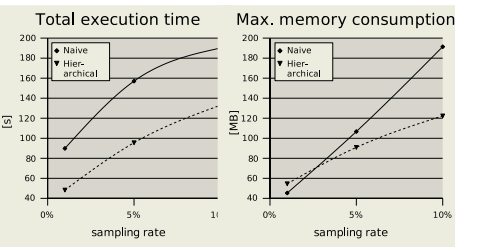

Fig. 7. Sampling rate & computation costs

Finally, the MFKJ of the base sample and the SGTs has to be computed. Since a tuple may appear in more than one sample table, it is worth computing the join before writing the sample tables in order to avoid unnecessary effort. Again, the discussion of several optimizations as well as sample maintenance is postponed to the full paper [8].

5 Evaluation

The hierarchical sampling techniques have been prototypically implemented and compared to the naive approach. The database (IBM UDB v8.1) has been addressed by a Java middleware, in which the sampling techniques have been integrated. The naive techniques have been implemented by using a view, and the hierarchical techniques have been implemented as described in the previous sections. The test system has been an AMD Athlon^{TM}XP 3000+ with 2 GB main memory. All tests have been executed with the TPC-D benchmark [13] and artificially skewed data. The size of the processed data is expressed by a scale factor. The relations *Nation* and *Region* have been excluded, since they only contain few tuples and do not scale. The skewness of the data has been simulated by a Zipf distribution with Zipf factor z. The Zipf factor $z = 1$ represents a uniform data distribution. A higher value of z yields more skewed data.

For an evaluation of the quality of the hierarchical senate sample, we used *Customer.Nationkey* and *Part.Type* as grouping attributes. The sampling rate has been 5%, the Zipf factor $z = 1.5$. Figure 6 compares the computation time of the sample and the main memory requirements. For large amounts of data, the hierarchical approach requires about 40% of the time the naive approach takes. The main memory requirements are almost identical for both approaches, even though the hierarchical approach uses additional data structures. The reason lies in the different use of the temporary reservoirs: on the one hand, they only consist of tuples from the source relation (hierarchical); on the other hand, these tuples are joined with all referenced relations in advance, and thus, become much bigger (naive).

Figure 7 depicts the influence of the sampling rate on the computation time and the memory requirements. The scale factor 0.1 has been used. The hierarchical approach accelerates the naive approach by a constant amount, i.e. it is

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

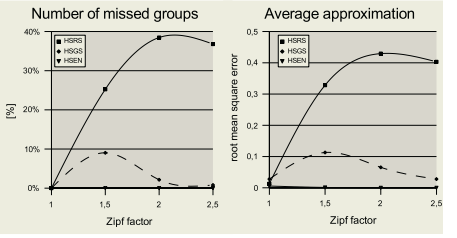

Fig. 8. Data skew & accuracy

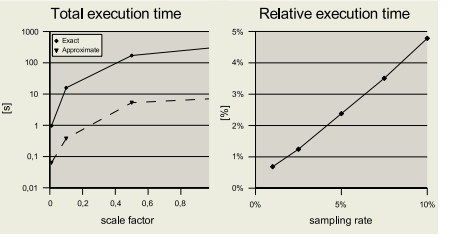

Fig. 9. Approximate computation costs

mostly independent from the sample size. However, the naive approach needs considerably more main memory with increasing sampling rates.

The same measurements have also been done for hierarchical small group sampling. The results are almost identical. The only difference is the fact that small group sampling needs considerably less main memory space both in the naive and the hierarchical variant, but requires more computation time instead. Due to space restrictions, these results are not shown in here.

As can be seen in Figure 8 (left), group-based sampling decreases the amount of non-recognized groups considerably. A grouping by *Customer.Nationkey* and *Supplier.Nationkey* has been done, the scale factor has been set to 0.1, and the metrics from [3] have been used. For a uniform data distribution ($z = 1$), all three sampling techniques offer comparable quality. But with increasing data skew, simple join synopses (HSRS) lose almost all small groups, while the HSGS only misses a few. The higher the data skew, the fewer middle-sized groups exist and the better small group sampling works. Finally, the HSEN recognizes all groups. It draws an advantage from the fact that it knows the grouping attributes in advance. The accuracy of the different techniques (cf. Figure 8, right) is evaluated similarly. For each group, the average revenue (price minus discount) has been calculated. Its root mean square error is shown in the figure.

Figure 9 (left) depicts the speed increase by approximate query processing with a sample size of 5% (logarithmic scale units). A scale factor of 0.1 has been used. The response time of join synopses and hierarchical group-based techniques is identical. On average, it is about 2.5% of the time required to compute the exact answer. As can be seen in Figure 9 (right), the sampling rate has a linear effect on the response time of an approximate query, relative to the one of an exact query.

6 Summary

We have shown how to efficiently combine foreign-key joins and group-based sampling. The resulting samples can be used to approximately answer queries, in which a relation and the relations referenced by it via foreign-key relationships are joined and/or in which groupings appear. It is not necessary to access the base data; all the required information is present in the sample.

Final edited form was published in "Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases. Sunderland 2005", S. 120-132, ISBN 978-3-540-31677-0
https://doi.org/10.1007/11511854_10

Our approaches can also be applied to other sampling techniques. For example, with the help of reference tables, the outliers detected by outlier indexing can be determined more efficiently.

Acknowledgement

This work has been supported by the German Research Society (DFG) under LE 1416/3-1.

References

1. University of California at Berkeley: How much Information? (2003) `http://www.sims.berkeley.edu/research/projects/how-much-info-2003/`.
2. Acharya, S., Gibbons, P., Poosala, V.: Congressional Samples for Approximate Answering of Group-By Queries. In: Proc. ACM SIGMOD. (2000) 487–498
3. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: Proc. ACM SIGMOD. (2003) 539–550
4. Acharya, S., Gibbons, P., Poosala, V., Ramaswamy, S.: Join synopses for approximate query answering. In: Proc. ACM SIGMOD. (1999) 275–286
5. Barbará, D., DuMouchel, W., Faloutsos, C., Haas, P., Hellerstein, J., Ioannidis, Y., Jagadish, H., Johnson, T., Ng, R., Poosala, V., Ross, K., Sevcik, K.: The New Jersey Data Reduction Report. IEEE Data Eng. Bull. **20** (1997) 3–45
6. Hellerstein, J., Haas, P., Wang, H.: Online Aggregation. In: Proc. ACM SIGMOD. (1997) 171–182
7. Vitter, J.: Random Sampling with a Reservoir. ACM Transactions on Mathematical Software **11** (1985) 37–57
8. Gemulla, R., Lehner, W.: On Incremental Maintenance of Materialized Offline Samples (2005) Submitted for publication.
9. Ganti, V., Lee, M., Ramakrishnan, R.: ICICLES: Self-Tuning Samples for Approximate Query Answering. In: The VLDB Journal. (2000) 176–187
10. Chaudhuri, S., Das, G., Datar, M., Motwani, R., Narasayya, V.: Overcoming Limitations of Sampling for Aggregation Queries. In: Proc. ICDE. (2001) 534–544
11. Chaudhuri, S., Motwani, R., Narasayya, V.: On Random Sampling over Joins. In: Proc. ACM SIGMOD. (1999) 263–274
12. Gemulla, R., Berthold, H., Lehner, W.: Hierarchical Group-based Sampling (2005) Full version available at `http://wwwdb.inf.tu-dresden.de/files/team/gemulla/files/hgs-fullversion.pdf`.
13. Transaction Processing Performance Council: TPC-D Benchmark Version 2.1. (1998) `http://www.tpc.org`.