Optimized Stratified Sampling for Approximate Query Processing

SURAJIT CHAUDHURI
Microsoft Research
and
GAUTAM DAS
University of Texas at Arlington
and
VIVEK NARASAYYA
Microsoft Research

The ability to approximately answer aggregation queries accurately and efficiently is of great benefit for decision support and data mining tools. In contrast to previous sampling-based studies, we treat the problem as an optimization problem where, given a workload of queries, we select a stratified random sample of the original data such that the error in answering the workload queries using the sample is minimized. A key novelty of our approach is that we can tailor the choice of samples to be robust even for workloads that are "similar" but not necessarily identical to the given workload. Finally, our techniques recognize the importance of taking into account the variance in the data distribution in a principled manner. We show how our solution can be implemented on a database system, and present results of extensive experiments on Microsoft SQL Server that demonstrate the superior quality of our method compared to previous work.

Categories and Subject Descriptors: H. [**Information Systems**]; H2. [**Database Systems**]
General Terms: Algorithms, Design, Experimentation, Performance
Additional Key Words and Phrases: Random Sampling, Approximation, Query Processing

1. INTRODUCTION

In recent years, decision support applications such as On Line Analytical Processing (OLAP) and data mining for analyzing large databases have become popular. A common characteristic of these applications is that they execute aggregation queries on large databases, which can often be expensive and resource intensive. Therefore, the ability to obtain approximate answers to such queries accurately and efficiently can greatly benefit the scalability of these applications. There have been several previous efforts to address this problem, ranging from using small random samples of the data, to other forms of data reduction techniques.

Authors' addresses: Authors 1 and 3, Microsoft Research, One Microsoft Way, Redmond WA 98052, {surajitc, viveknar}@microsoft.com; Author 2, Department of Computer Science and Engineering, The University of Texas at Arlington., Arlington TX 76019, gdas@cse.uta.edu. Part of this work was performed while Author 2 was a researcher at Microsoft Research. A conference version of this paper titled "A robust, optimization-based approach for approximate answering of aggregate queries" appeared in SIGMOD 2001. This is a preliminary release of an article accepted by ACM Transactions on Database Systems. The definitive version is currently in production at ACM and, when released, will supersede this version.

In this paper we exclusively focus on the approach of using *pre-computed samples* of the data to answer the queries. A seemingly simple and efficient technique for approximate query processing is to use a pre-computed uniform random sample for answering queries. However, uniform random sampling has two well-known disadvantages. First, avoiding large errors on an arbitrary query, especially for queries with relatively low selectivity, is virtually impossible. This is because the uniform sample may not adequately represent all the data tuples that are selected by the query. Second, uniform random sampling ignores the variance in the data distribution of the aggregated column(s). As the following example shows, ignoring data variance can lead to poor quality of answers for aggregate functions such as SUM:

Example 1. Consider a relation R containing two columns <*ProductId*, *Revenue*> and four records {<1, 10>, <2, 10>, <3, 10>, <4, 1000>}. Assume that we are allowed to use a sample S of two records from R to answer the query Q: SELECT SUM(Revenue) FROM R. We answer a query by running it against S and scaling the result by a factor of two (since we are using a 50% sample). Consider a sample S_1 = {<1, 10>, <3, 10>}. The estimated answer for Q using S_1 is 40, which is a severe underestimate of the actual answer (1030). Now consider another sample S_2 = {<1, 10>, <4, 1000>}. The estimated answer for Q using S_2 is 2020, which is a significant overestimate. Thus, large variance in the aggregate column can lead to large relative errors.

Previous studies have proposed using the *workload* to guide the process of selecting samples to minimize the effects of the first problem of low selectivity (Acharya et al [2000]; Ganti et al [2000]). The hope is that by picking a sample that is tuned to the given workload, we can ensure acceptable error at least for queries in the workload. The problem of large data variance has been addressed in Chaudhuri et al [2001], who proposed that records with outlier values (i.e., values that significantly deviate from the mean) be organized in a separate *outlier index*, and a uniform random sample be constructed out of the remaining data. However, despite recognizing the importance of the above two problems with uniform random sampling, previous studies suffer from several significant drawbacks. First, although the proposed solutions have intuitive appeal, the lack of a rigorous problem formulation leads to suboptimal solutions that are difficult to evaluate theoretically. Second, the solutions address each problem orthogonally, i.e., they do not attempt to develop a unifying approach framework to solve both problems together. Third, they do not attempt to formally deal with uncertainty in the expected workload, i.e., when incoming queries are “similar” but not identical to queries in the given workload.

In contrast to most previous sampling-based studies, in this paper we propose a more principled approach by formulating the problem of pre-computing a sample as an *optimization problem*, whose goal is to pick a sample that minimizes the error for the given workload. In fact, we introduce a generalized model of the workload, which we refer to as a *lifted workload*, that makes it possible to tune the choice of the sample so that approximate query processing using the sample is effective not only for workloads that are identical to the given workload, but also for the lifted workload model – i.e., for workloads that are “similar” to the given workload in the sense the queries select regions of the data that overlap significantly with the data accessed by the queries in the given workload. We formulate selection of the sample for such a workload as a *stratified sampling problem* with the goal to minimize error in estimation of aggregates. Our formulation makes the problem amenable to exploiting known techniques in stratified sampling and optimization.

As a consequence, we have developed a robust algorithm, called *STRAT*, for the problem of approximate query processing of queries with selections, foreign-key joins, GROUP BY and aggregation operators such as COUNT, SUM, and AVG. Table 1 summarizes the types of queries that our approximate query answering system can answer.

Selection	Arbitrary (single block) selection conditions
Join	Star queries with foreign key joins (see Section 5.3)
Group By	yes
Aggregation	COUNT, SUM, AVG

Table 1. Types of SQL queries answerable by STRAT

We have implemented our solutions on Microsoft SQL Server 2000, addressing the pragmatic issues that are central to an effective solution that can be deployed in a commercial DBMS. The benefits of our systematic approach are amply demonstrated not only by theoretical results, but also experimentally on synthetic as well as on a deployed enterprise data warehouse in our organization.

We present some details of our proposed architecture for approximate query processing, summarized in Figure 1. The inputs are a database and a workload **W.** For simplicity, we present the case where the database is a single relation **R**. As with previous sampling-based studies, we have taken the approach of exploiting the available workload to find samples that work well for queries in the given workload. A workload **W** is specified as a set of pairs of queries and their corresponding weights: i.e., **W** = {<Q_1, w_1>, … <Q_q, w_q>}, where weight w_i indicates the importance of query Q_i in the workload. Without loss of generality, we can assume that the weights are normalized, i.e., $\Sigma_i w_i = 1$. In practice, such a workload may be obtained using profiling tools available on most modern DBMSs for logging queries that execute on the server.

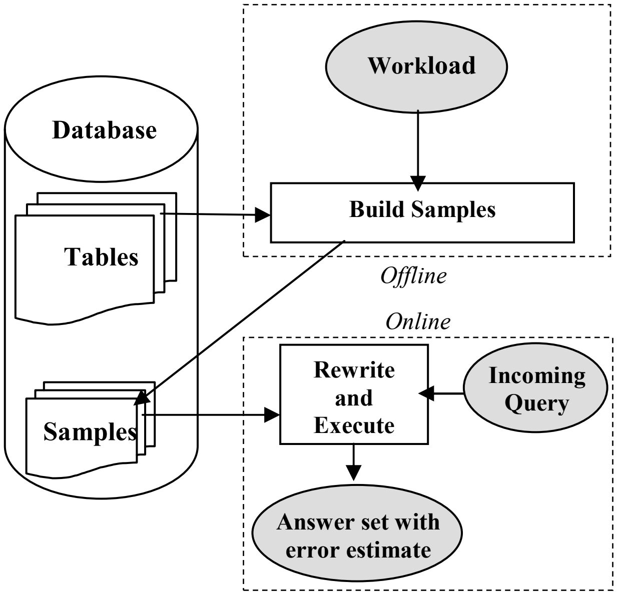


Figure 1. Architecture for Approximate Query Processing

There are two components in our architecture: (1) an *offline* component for selecting a sample of records from relation **R**, and (2) an *online* component that (a) rewrites an incoming query to use the sample (if appropriate) to answer the query approximately and (b) reports the answer with an estimate of the error in the answer.

Consider the offline component, "Build Samples". We assume that a pre-designated amount of storage space is available for selecting samples from the database. The algorithm STRAT samples records using a randomization technique based on a generalized model of the workload. The latter case is a significant contribution of this paper since it is unrealistic to assume that incoming queries in the future will be identical to queries present in the given workload. We present a method for automatically "lifting" a given workload, i.e., we compute a *probability distribution* $p_{\mathbf{W}}$ of incoming queries, where $p_{\mathbf{W}}(Q)$ is the probability that the next incoming query is Q - the subscript indicates that the distribution depends on **W**. Effectively, $p_{\mathbf{W}}(Q)$ represents a generalized model of the workload. Our algorithm then selects - using stratified sampling techniques - a sample that is resilient enough for such a lifted workload. Furthermore, this sample is optimal, in that it minimizes the expected error of answering queries in the (lifted) workload.

In the online component, an incoming query is rewritten to run against the samples instead of the base relation. For a multi-relation query, in addition to the samples, we may also reference other base relations to answer the query. As in previous work (Acharya et al [2000]; Chaudhuri et al [2001]; Ganti et al [2000]), we assume that each record in the sample also contains an additional column known as the *ScaleFactor* with each record. The value of the aggregate column of each record in the sample is first scaled up by multiplying with the *ScaleFactor*, and then aggregated. In addition to the approximate answer, we can also report the variance (or even a confidence interval) for the approximate answer. The online component is based on standard query rewriting techniques developed in previous works (Acharya et al [2000]; Chaudhuri et al [2001]; Ganti et al [2000]) and thus we do not focus on it much in this paper - the novelty of this paper is in the first component.

The rest of this paper is organized as follows. We present preliminary concepts from classical sampling theory in Section 2. We describe a model for lifting a given workload in Section 3. We present our main algorithm STRAT in Section 4. We describe extensions to STRAT for a broader class of queries in Section 5. We discuss sample maintenance issues in the presence of updates in Section 6. We discuss related work in Section 7, including a detailed analysis of the sub-optimality of previous sampling-based approaches. We describe our experimental results in Section 8, and conclude in Section 9. The proofs of several technical lemmas are presented in an appendix.

2. PRELIMINARIES

In this section we first present error metrics by which the quality of an approximate query processing system can be measured. We then present a brief overview of classical sampling theory that is used in designing our approximate query processing system.

2.1 Error Metrics for Approximate Query Answering

Consider a SQL query Q with a COUNT or SUM aggregate. Suppose the correct answer for a query Q is y while the approximate answer is y'. We focus on *relative error* instead of *absolute error*, since the former is usually a fairer measure across queries. Relative error is defined as $E(Q) = |y - y'|/y$. (However, for the sake of brevity, we henceforth omit the word "relative" in all references to error metrics.) The *squared error* is defined as $SE(Q) = ((y - y')/y)^2$. Now consider a GROUP BY query Q that induces g groups in the data. Suppose the correct answer for the ith group is y_i while the approximate answer is y_i'. The squared error in answering the query is $SE(Q) = 1/g \sum_i ((y_i - y_i')/y_i)^2$. This error measure for a GROUP BY query has also been considered by Acharya et al [2000] and Chaudhuri et al [2001]. In other words, a GROUP BY query can be treated as g SELECT queries, each of weight $1/g$. Given a probability distribution of queries p_W, the *mean squared error* for the distribution is defined as $MSE(p_W) = \sum_Q p_W(Q) SE(Q)$ where p_W (Q) is the probability of query Q. The *root mean squared error*, also known as the L_2 error, is defined as $RMSE(p_W) = \sqrt{MSE(p_W)}$. Other error metrics are possible e.g., using the L_1 metric (defined as the expected relative error over all queries) or L_∞ metric (defined as the maximum relative error over all queries). In this paper, although we optimize for the *MSE* due to its long tradition in statistics, we found that these solutions also do very well for the L_1 metric. Since most previous work in this area report the L_1 metric, our experimental comparisons also report the L_1 metric.

2.2 Classical Sampling Techniques

Many approaches to answering approximate answering of aggregation queries - including some of our own techniques presented later in this paper - are based on executing the query against a *random sample* of the database. The techniques primarily differ in the way the random sample is pre-computed, and most are adaptations of well-known techniques from classical sampling theory. We now review some of these classical sampling techniques. Much of these results are available in any basic book on sampling theory, such as Cochran [1977] and Lohr [1999]. Sampling techniques have traditionally been applied to estimate aggregates of a single given population (such as the average per capita income of a country). The reason sampling is used is because the population is usually too large to allow exact aggregate computation.

2.2.1 Uniform Sampling

Consider a large population, i.e. a set of real numbers R = {y_1,…, y_n}. Let the average be *y*, the sum be Y and the variance be S^2. Suppose we uniformly sample *k* numbers with replacement. Let the mean of the sample be μ.

Lemma 1 (Uniform Sampling): *(a) μ is an unbiased estimator for y, i.e.* $E[\mu] = y$*; (b)* $\mu \cdot n$ *is an unbiased estimator for Y, i.e.* $E[\mu \cdot n] = Y$*; (c) The variance (or standard error) in estimating y is* $E[(\mu - y)^2] = S^2 / k$*; (d) The variance in estimating Y is* $E[(\mu \cdot n - Y)^2] = n^2 S^2 / k$*; (e) The relative squared error in estimating Y is* $E\left[\left((\mu \cdot n - Y)/Y\right)^2\right] = n^2 S^2 / Y^2 k$.

The above squared error formulas assume that *n* is much larger than *k* (more accurate formulas that are sensitive to the finiteness of *n* are also known). As can be seen, the squared errors of the estimates depend directly on the variance in the data, and inversely on the number of samples. Also, the formulas are different if we perform random sampling *without* replacement (which is often the case in practical sampling systems such as ours); however, when *n* is much larger than *k* these differences are negligible.

2.2.2 Stratified Sampling

One can often do better than uniform sampling by exploiting available (partial) knowledge of the population. For example, information on last year's per capita income distribution may be available, and can be assumed to be highly correlated with this year's income distribution. One can identify regions of high variance in last year's data, and design a sampling strategy where one samples more from such regions. Such a scheme can be a highly accurate estimator for the current population. One such strategy is known as *stratified sampling*. Here the current population is partitioned into *r* strata, with the *j*th stratum R_j containing n_j numbers that have sum Y_j and variance S^2_j. Suppose we uniformly sample k_1,…, k_r numbers from each of the R_1,…, R_r strata respectively. Let the means of the respective samples be μ_1… μ_r. Define $\mu = \sum_j n_j \mu_j / n$.

Lemma 2 (Stratified Sampling): *(a) μ is an unbiased estimator for y, i.e.* $E[\mu] = y$*; (b)* $\mu \cdot n$ *is an unbiased estimator for Y, i.e.* $E[\mu \cdot n] = Y$*; (c) The variance in* *estimating y is* $E[(\mu - y)^2] = 1/n^2 \sum_j n_j^2 S_j^2 / k_j$ *; (d) The variance in estimating Y is* $E[(\mu \cdot n - Y)^2] = \sum_j n_j^2 S_j^2 / k_j$ *; (e) The relative squared error in estimating Y is* $E\left[((\mu \cdot n - Y)/Y)^2\right] = 1/Y^2 \sum_j n_j^2 S_j^2 / k_j$.

As with uniform sampling, the error formulas here assume that each n_j is much larger than the corresponding k_j. Stratified sampling can be better than uniform sampling because different strata can be designed to reduce the variance. The issues in stratified sampling are: how to select *r* (the number of strata), how to partition the population into *r* strata, how to allocate a total of *k* samples over all strata so as to minimize the error. If perfect information about the population is available, then more the strata, the better (only limited by the fact that *k* samples have to be distributed among *r* strata). Answering these questions require the availability of (partial) information about the current population.

Lemma 3 (Neyman Allocation): *Given a population* ***R*** *= {y₁,..., yₙ}, k and r, the optimal way to form r strata and allocate k samples among all strata is to first sort R and select strata boundaries so that* $\sum_j n_j S_j$ *is minimized, and then for the jth strata, to set the number of samples* k_j *as* $k_j = k\left(n_j S_j \Big/ \sum_j n_j S_j \right)$.

Given *r*, the above lemma shows how the population should be stratified to minimize the error. However, the lemma does not answer the question of how to select *r*. If we have complete knowledge of the current population, it is easy to see that the more strata the better. However we usually have only partial knowledge of the population (e.g. we may have last year's population distribution which does not fully correlate with this year's population), so stratified sampling starts getting inaccurate beyond *r* = 6 (Cochran [1977]). An efficient procedure described in Cochran [1977] to approximate the optimal stratification is as follows: if a density distribution of the population is available (say *f*(*y*)), compute the cumulative of the function $\sqrt{f(y)}$ and choose the strata boundaries so that they make equal intervals on this cumulative scale.

2.2.3 Multivariate Stratified Sampling

Now suppose the population R = $\{y_1,\ldots, y_n\}$is a set of m-dimensional vectors, i.e. each y_i = $[y_{i,1}, y_{i,2}, \ldots, y_{i,m}]$. Suppose we want to estimate the means along each dimension. Given *k* and *r*, we wish to form *r* strata and allocate *k* samples among all strata such the sum of the squared errors along each dimension is minimized. This multivariate generalization of the Neyman Allocation has been tackled in several works such as Bethel [1998] and Cochran [1977]. In our application we encounter a variation of this problem, which we solve using a simple clustering based heuristic (see Section 5.1).

2.2.4 Weighted Sampling

Weighted sampling can be viewed as an approximation of stratified sampling. Each number in the entire population has to be examined in order to select the sample. Assume that we are given a parameter *k*, and that each y_t in the population has been assigned a weight w_t (indicating its importance). Each y_t is selected to be included in the sample with probability $k\left(w_t \Big/ \sum_u w_u \right)$. When weights are the same, this reduces to an approximation of uniform sampling.

2.2.5 Error Estimations and Confidence Intervals

Often when sampling techniques are used to estimate the mean of a population, it is desirable to output an estimate of the error (or variance) in the answer in addition to the sample mean. However, if we examine the formulas for the error, e.g., Lemma 1(c), we notice that it requires knowledge of the variance of the whole population, which may not be known. Nevertheless, a good estimate of the error can be obtained by replacing the population variance with the sample variance in the formulas; the latter is of course computable from the sample. For large sample sizes, it is well known (by the Central Limit Theorem) that the distribution of the sample mean approaches the normal distribution. This fact allows us to leverage properties of normal distributions to also output *confidence intervals* for the sample mean – e.g., we may be able to say that the population mean is within $[\mu\text{-}\varepsilon, \mu+\varepsilon]$ with high probability.

In the rest of this paper we omit further discussion of error estimates and confidence intervals, but remark that our techniques for approximate query processing can be extended in a straightforward manner to output error estimates and/or confidence intervals in addition to approximate answers of aggregation queries. As a final point, we note that the above theory assumes that a fresh random sample is drawn to answer every query. However, the majority of approximate query processing systems (including ours) assume the construction of a pre-computed sample that is repeatedly re-used for multiple queries. This poses potential problems when a stream of queries are correlated and the same sample is used to answer them. In practical terms, a sample can become "stale," and the effects of obtaining an inferior sample (due to bad luck of the draw) will extend over many queries. Even within the same GROUP BY query, different groups are highly correlated – e.g., groups have to be disjoint from one another – thus estimating the errors for different groups in the same query using the same sample is not as simple as described above. Correcting for such situations will require the adoption of techniques of *simultaneous statistical inference* (Miller [1981]). However, we leave such efforts for future work - in this paper (as discussed in the next section) we only assume that the incoming queries are independently drawn from a fixed distribution.

3. LIFTING WORKLOAD TO QUERY DISTRIBUTIONS

As mentioned earlier, we would like our approximate query processing scheme to not only perform well for incoming queries that exactly match one of the queries in the given workload, but also be resilient to the situation when an incoming query is "similar" but not identical to queries in the workload. In this section we tackle one aspect of the problem, i.e., defining this notion of similarity. More formally, we show how given **W**, we can define a *lifted workload* $p_{\mathbf{W}}$, i.e., a probability distribution of incoming queries. Intuitively, for any query Q' (not necessarily in **W**), $p_{\mathbf{W}}$(Q') should be related to the amount of similarity (dissimilarity) of Q' to the workload: high if Q' is similar to queries in the workload, and low otherwise. In Sections 4 and 5 we show how to leverage such a probability distribution in our approximate query processing solution.

Our notion of similarity between queries is not concerned with syntactic similarity of query expressions. Rather, we say that two queries Q' and Q are similar if the records selected by Q' and Q have significant overlap. We focus on the case of single-table selection queries with aggregation containing either the SUM or COUNT aggregate (this intuition is refined for GROUP BY and join queries in Section 5). Let us consider the simplest case when the workload **W** consists of exactly one query Q on relation R. Let R_Q be the records selected by Q. Our objective is to define the distribution $p_{\{Q\}}$ (i.e., for $p_{\mathbf{W}}$, where **W** = {<Q,1.0>}). Since for the purposes of lifting, we are only concerned with the set of records selected by a query and not the query itself, we make a change in notation for convenience: instead of mapping queries to probabilities, $p_{\{Q\}}$ maps subsets of R to probabilities[3]. For all R'⊆R, $p_{\{Q\}}$(R') denotes the probability of occurrence of any query that selects exactly the set of records R'.

For the moment, assume two parameters δ and γ have been specified such that $0 \leq \gamma, \delta \leq 1$. Informally, these parameters define the degree to which the workload "influences" the query distribution. More formally, for any given record inside (resp. outside) R_Q, the parameter δ (resp. γ) represents the probability that an incoming query will select this record.

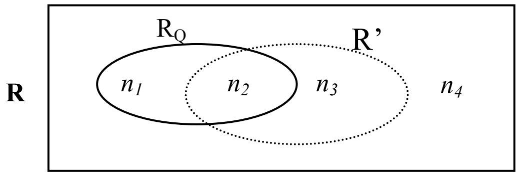


Figure 2

Given these two parameters, we can now derive $p_{\{Q\}}$(R') for any R'⊆ R (i.e. the probability of occurrence of any query that exactly selects R'). Figure 2 shows a Venn diagram of R, R_Q and R', where n_1, n_2, n_3, and n_4 are the counts of records in the regions indicated. Equation 1 shows the derivation of $p_{\{Q\}}$(R'). Note that when n_2 or n_4 are large (i.e., the overlap is large), $p_{\{Q\}}$(R') is high (i.e. queries that select R_Q are likely to occur), whereas when n_1 or n_3 are large (i.e. the overlap is small), $p_{\{Q\}}$(R') is low (i.e. queries that select R_Q are unlikely to occur). Once $p_{\{Q\}}$ has been defined, $p_{\mathbf{W}}$ can be easily derived, as shown in Equation 2.

$$p_{\{Q\}}(R') = \delta^{n_2}(1-\delta)^{n_1}\gamma^{n_3}(1-\gamma)^{n_4} \qquad (1)$$

$$p_{\mathrm{W}}(R') = \sum_{i=1}^{q} w_i p_{\{Q_i\}}(R') \qquad (2)$$

Let us now discuss the problem of setting the parameters δ and γ. As mentioned earlier, the parameters define the degree to which the workload **W** influences the query distribution $p_{\mathbf{W}}$. Table 2 elaborates on this issue by analyzing the effects of different boundary settings of these parameters.

[3] This notation makes it convenient to give a single probability to the (infinite) set of queries that only syntactically differ in their WHERE clauses, yet select the same R'. Note that the domain of $p_{\{Q\}}$ is finite, i.e. the power set of R.

Boundary Settings	**Interpretation**
$\delta = 1$ and $\gamma = 0$	incoming queries are identical to workload queries
$\delta = 1$ and $\gamma > 0$	incoming queries are supersets of workload queries
$\delta < 1$ and $\gamma = 0$	incoming queries are subsets of workload queries
$0 \leq \gamma = \delta \leq 1$	incoming queries (with expected selectivity δ (= γ)) are uncorrelated with the workload queries

Table 2. Interpretation of different boundary settings of parameters δ and γ

In general, γ and δ will fall somewhere in between, and using the above scenarios as guidelines, it may be possible for skilled database administrators to analyze their workload patterns, and manually set the parameters to values that best model their workloads. However, we also present a simple automated approach for parameter setting. The basic idea is to split the available workload into two sets, the *training workload* and the *test workload*. The parameters are selected using a two-dimensional grid search approach (based on Valliant [1997]) such that the lifted training workload (under these settings) most closely fits the test workload. Essentially we divide the two-dimensional space $0 \leq \delta, \gamma \leq 1$ into a grid in which each dimension is divided into a fixed number of intervals. For each point (δ, γ) in the grid, we compute a sample for the training set and estimate the error for the test set. We pick the grid point with the lowest error for the test set as our setting for δ and γ. This grid search approach is effective and scalable with data size for low dimensional optimization problems such as ours since we can obtain samples for multiple grid points in one scan of the relation, and our experiments (Section 8) indicate the approach is promising. We are also investigating alternative approaches such as randomized search and gradient descent.

The above represents a simple first attempt at lifting workloads in a rigorous manner. Other methods for lifting a workload need to be studied in the future, and in fact, the problem of lifting a workload is really orthogonal to the problem of approximate query processing, and we expect it to find applications in other areas.

In the next few sections, we develop an approximate query processing scheme called STRAT, which will attempt to minimize the *MSE* of the lifted workload, i.e., for $p_{\mathbf{W}}$.

4. THE STRAT ALGORITHM

In this section, we present a formulation of the approximate query processing problem and our stratified sampling based solution, STRAT. For simplicity, we discuss the case where the database consists of a single relation R. The formulation can be extended for multi-table queries (see Section 5).

4.1 Problem Formulation

***Problem*: AQP**
Input: R, $p_{\mathbf{W}}$ (a probability distribution function specified by **W**), and *k*
Output: A sample of *k* records, (with the appropriate additional column(s)) such that the *MSE*($p_{\mathbf{W}}$) is minimized

In the above formulation, $p_{\mathbf{W}}$ is any probability distribution function derived from the given workload **W**. For example, the lifting model presented in Section 3 can be used to obtain $p_{\mathbf{W}}$.

Before we give the formal details of STRAT, we give some intuition to justify the rationale for stratified sampling.

4.2 Rationale for Stratified Sampling

As discussed in Section 2, stratified sampling is a well-known generalization of uniform sampling where a population is partitioned into multiple strata and samples are selected uniformly from each stratum, with "important" strata contributing relatively more samples.

Consider the following selection query with aggregation on relation R defined in Example 1 in the introduction. Q_1 = SELECT COUNT(*) FROM R WHERE *ProductId* IN (3,4). Recall that R is the relation {<1, 10>, <2, 10>, <3, 10>, <4, 1000>}. We define the *population* of a query Q (denoted by POP_Q) on a relation R as a set of size |R| that contains the value of the aggregated column that is selected by Q, or 0 if the record is not selected. By this definition, POP_{Q1} = {0, 0, 1, 1}. Observe that POP_{Q1} has a mix of 1's and 0's and thus has a non-zero variance. Thus, a uniform sampling of POP_{Q1} would be a poor choice for this problem since it would incur non-zero error. However, if we partition R into two *strata* {<1, 10>, <2, 10>} and {<3, 10>, <4, 1000>}, we effectively partition POP_{Q1} into two strata {0, 0} and {1, 1}. Each stratum now has zero variance, and a stratified sampling strategy that selects at least one sample from each stratum will estimate Q_1 with zero error.

Note however, that this particular stratification may not work well for a different COUNT query whose population has a different distribution of 1s and 0s. For example, consider a query Q_2 = SELECT COUNT(*) FROM R WHERE *ProductId* IN (1,2,3). POP_{Q2} = {1, 1, 1, 0} and is different from POP_{Q1}. As can be seen by this example, each query defines its own population of the same relation R, and therefore the challenge is to adapt stratified sampling so that it works well for all queries. An effective scheme will need to stratify the relation such that the expected variance over all queries in each stratum is small, and allocate more samples to strata with larger expected variances.

For SUM queries, stratification is also governed by the additional problem of variance in the aggregate column. For example, consider query Q_3 = SELECT SUM(*Revenue*) FROM R WHERE *ProductID* IN (1,4). $POP_{Q3} = \{10, 0, 0, 1000\}$ and therefore has large variance.

Thus, in general a stratified sampling scheme partitions R into r strata containing n_1, ., n_r records (where $\Sigma n_j = n$), with $k_1, \ldots, k_r$ records uniformly sampled from each stratum (where $\Sigma k_j = k$). As mentioned in the introduction, the scheme also associates a *ScaleFactor* with each record in the sample. Queries are answered by executing them on the sample instead of R. For a COUNT query, the *ScaleFactor* entries of the selected records are summed, while for a SUM(y) query the expression y*ScaleFactor* is summed. If we also wish to return an error guarantee with each query, then instead of *ScaleFactor*, we have to keep track of each n_j and k_j individually for each stratum.

We now formally present STRAT for queries containing the COUNT aggregate, and in Section 4.4 we describe the extensions necessary to deal with the more challenging SUM aggregate. (Further extensions to STRAT to handle joins and GROUP BY are deferred to Section 5).

4.3 Solution for COUNT Aggregate

Our solution consists of three steps. The first step, which we refer to as *stratification*, is determining (a) how many strata r to partition relation R into, and (b) the records from R that belong to each stratum. At the end of this step we have r strata $R_1, \ldots R_r$ containing $n_1, \ldots n_r$ records such that $\Sigma n_j = n$. The second step, called *allocation*, determines how to divide k (the number of records available for the sample) into integers $k_1, \ldots, k_r$ across the r strata such that $\Sigma k_j = k$. The third step, referred to as the *sampling* step, uniformly samples k_j records from stratum R_j to form the final sample of k records. The sample so created is then used at runtime to approximately answer queries. The heart of the algorithm is in the first two steps, which are designed to minimize the errors in approximately answering queries in the lifted workload ($p_{\mathbf{W}}$). The third step is straightforward, and can be accomplished with one scan of relation R using reservoir sampling techniques (Fan et al [1962]; Vitter [1985]).

4.3.1 Stratification

It may appear that the problem of stratification of R for a given workload W of COUNT queries is intractable since when *r* is not known, there are an exponential number of ways of stratifying R. To alleviate this problem, we introduce the concept of *fundamental regions*. For a given relation R and workload **W**, consider partitioning the records in R into a minimum number of regions F = {R_1, R_2, …, R_r} such that for any region R_j, each query in **W** selects either all records in R_j or none. These regions are the fundamental regions of R induced by **W**. For example, consider a relation R (with aggregate column C) containing nine records (with C values 10, 20, …, 90), as shown in Figure 3. Let **W** consist of two queries, Q_1 (which selects records with C values between 10 and 50) and Q_2 (which selects records with C values between 40 and 70). These two queries induce a partition of R into four fundamental regions, labeled R_1,…,R_4.

The concept of *finest partitioning* into groups in Acharya et al [2000] is similar to the concept of fundamental regions. In general the total number of fundamental regions *r* depends on R and **W** and is upper-bounded by min($2^{|\mathbf{W}|}$, *n*) where *n* is the number of records in R. The algorithmic and implementation details of how to identify fundamental regions efficiently are discussed in Section 4.5.

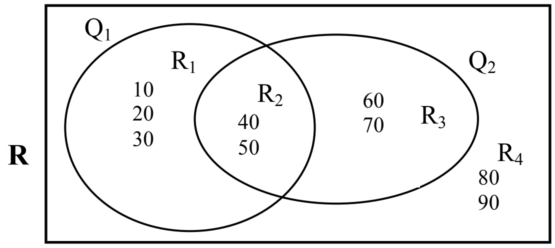


Figure 3. Fundamental Regions

Under *large population assumption* (i.e., when *n*, the number of records in R, is large), the following lemma says that it is enough to partition R into fundamental regions and treat each region as a stratum.

Lemma 4: *Consider a relation R with n records and a workload* ***W*** *of COUNT queries. In the limit when n tends to infinity, the fundamental regions F = {R_1, R_2, ..., R_r} represent an optimal stratification.*

The proof of this lemma (as well as some of the other more involved lemmas to follow in this section) has been deferred to the appendix.

4.3.2 Allocation

Once the stratification has been done, the key remaining challenge is how to allocate the *k* records across the *r* fundamental regions (strata). Our main idea is to treat this problem as an optimization problem whose goal is to minimize the error over queries in $p_{\mathbf{W}}$. Observe that this is a *significant* point of departure compared to most previous work in this area, where this allocation step is done in an intuitive but informal manner. We assume that $k_1,\ldots, k_r$, are unknown variables such that $\Sigma k_j = k$. We leverage the following two results to express $MSE(p_{\mathbf{W}})$ as a function of these variables and then select values for these variables that minimizes $MSE(p_{\mathbf{W}})$. First, using Equation 2 (Section 3), it is easy to see that $MSE(p_{\mathbf{W}})$ can be expressed as a weighted sum of the *MSE* of each query in the workload (as stated by the following lemma).

Lemma 5: $MSE(p_W) = \sum_{i=1}^{q} w_i MSE(p_{\{Q_i\}})$

Next, for any Q ∈ **W**, we express $MSE(p_{\{Q\}})$ as a function of the k_j's. Although obtaining a concise yet exact expression for this function is more difficult, under *large population assumption* the following lemma (one of the principal results of this paper) shows how to obtain a succinct approximation for $MSE(p_{\{Q\}})$. In our experiments, we have found that this formula for $MSE(p_{\{Q\}})$ has yielded excellent approximations for realistic values of *n*.

Lemma 6: *For a COUNT query Q in **W**, let*

$$ApproxMSE(p_{\{Q\}}) = \frac{\sum_{R_j \subseteq R_Q} \frac{n_j^2}{k_j}\delta(1-\delta) + \sum_{R_j \subseteq R \setminus R_Q} \frac{n_j^2}{k_j}\gamma(1-\gamma)}{\left(\sum_{R_j \subseteq R_Q} \delta n_j + \sum_{R_j \subseteq R \setminus R_Q} \gamma n_j\right)^2}$$

Then $\lim_{n\to\infty} \frac{MSE(p_{\{Q\}})}{ApproxMSE(p_{\{Q\}})} = 1$

Outline of Proof: We provide an outline of the proof for the case where we assume *each* n_j is large - the proof for the more general case where we assume only *n* to be large appears in the appendix. Let Q' be a query randomly drawn from the distribution $p_{\{Q\}}$. The number of records selected by Q' in each fundamental region follows a binomial distribution. Since each n_j is large, an overwhelming number of queries from the distribution $p_{\{Q\}}$ will select approximately $\delta.n_j$ (resp. $\gamma.n_j$) records from R_j, where R_j is a fundamental region inside (resp. outside) R_Q. Thus, $MSE(p_{\{Q\}})$ can be approximated as the MSE of all such queries since the contribution from the other queries is negligible. Consider the j*th* term in the left summation in the numerator. It represents the expected squared error in estimating the count of $(R_{Q'} \cap R_j)$. Similarly, the right summation in the numerator represents the expected squared error in estimating the count of $(R_{Q'} \cap (\mathbf{R} \backslash R_Q))$. Thus the numerator represents the expected squared error in estimating the count of $R_{Q'}$. Dividing by the denominator represents the expected *relative* squared error in estimating the count of $R_{Q'}$. ■

Now that we have an approximate formula for $MSE(p_{\{Q\}})$, we can also define an approximate formula for $MSE(p_W)$. Analogous to the formula in Lemma 5, let $ApproxMSE(p_W) = \sum_{i=1}^{q} w_i ApproxMSE(p_{\{Q_i\}})$. Using Lemma 6, it is easy to see that $\lim_{n \to \infty} \frac{MSE(p_W)}{ApproxMSE(p_W)} = 1$. Moreover, the following corollary to Lemma 6 shows that we can express $ApproxMSE(p_W)$ as a function of the variables $k_1, \ldots, k_r$.

Corollary 1: *Let* $\alpha_{j,Q} = \frac{\sum_{R_j \subseteq R_Q} n_j^2 \delta(1-\delta) + \sum_{R_j \subseteq R \backslash R_Q} n_j^2 \gamma(1-\gamma)}{\left(\sum_{R_j \subseteq R_Q} \delta n_j + \sum_{R_j \subseteq R \backslash R_Q} \gamma n_j \right)^2}$

and $\alpha_j = \sum_{i=1}^{q} w_i \alpha_{j,Q_i}$. *Then* $ApproxMSE(p_W) = \sum_{1 \le j \le r} \alpha_j / k_j$.

Proof: Since $ApproxMSE(P_{\{Q\}}) = \sum_{1 \le j \le r} \alpha_{j,Q} / k_j$ (Lemma 6), summing this over all queries in the workload yields the corollary. ■

Intuitively, α_j captures the "importance" of a region; it is positively correlated with n_j as well as the frequency of queries in the workload that access R_j. Once we have approximately expressed $MSE(p_W)$ as a function of the unknown k_j's, we are ready to minimize it.

Lemma 7: $\sum_{1 \le j \le r} \alpha_j / k_j$ is minimized under the constraint $\sum_{1 \le j \le r} k_j = k$ by setting

$$k_j = k\left(\sqrt{\alpha_j} \Big/ \sum_{1 \le i \le r} \sqrt{\alpha_i} \right)$$

The proof of Lemma 7 has been deferred to the appendix. Note that Lemma 7 provides us with a closed-form and computationally inexpensive solution to the allocation problem since α_j depends only on δ, γ and the number of records in each fundamental region. Since an admissible solution in our case requires that each k_j is an integer, we round each k_j to the nearest integer. For now, we assume that STRAT completes its allocation by dividing k into $k_1,\ldots,k_r$ according to Lemma 7.

4.4 Solution for SUM Aggregate

We now highlight the extensions to the above solution required for queries containing only the SUM aggregate. The key difference arises due to the fact that for SUM, we also need to take into account the *variance* of the data in the aggregated column (see Example 1 in Section 1). The first effort to deal with variance in data for approximate query processing was the *outlier-indexing* technique presented in Chaudhuri et al [2001]. We use a more general and principled approach that adapts techniques from statistics for dealing with large variance. We note that both the stratification and allocation steps for the SUM are sufficiently different from COUNT, and need to be revisited.

Before we get into the details of our solution for SUM, we discuss an interesting scenario in which our solution (and in fact, most sampling based solutions) will fail to work. Consider a relation R that has a mix of positive and negative numbers, and furthermore suppose a subset R' exists whose SUM is close to zero (i.e. the negative values cancel the positive values), but whose variance is large. Even though a query Q' that selects R' may have a small probability of occurrence in the lifted distribution, if not answered exactly, its relative error can become infinite. Most sampling methods cannot handle such queries, and these queries need to be recognized and processed separately.[4]

As we shall show, this problem does not arise if the values in R are all strictly positive or strictly negative. The solution that we present is optimized only for such databases, and in principle can fail for certain kinds of queries on more general databases. However,

[4] An alternative *hybrid error metric* can alleviate this problem, where the error of a query is the defined as the absolute error when the query aggregate is below a certain threshold, and as the relative error when the aggregate is above this threshold. Deriving optimal sampling strategies for such hybrid error metrics is left for future work.

we note that in our experiments our solution has worked consistently well for all kinds of databases.

4.4.1 Stratification

If we use the same stratification as in the COUNT case, i.e., strata = fundamental regions, we may get poor solutions for SUM since each stratum now may have large internal variance in the values of the aggregate column. Therefore, we use a bucketing technique where we further divide fundamental regions with large variance into a set of *finer regions*, each of which has significantly lower internal variance. We then treat these finer regions as the strata.

For the step of further dividing a fundamental region, we borrow from statistics literature an approximation algorithm for the optimal *Neyman Allocation* technique (see Lemma 3 in Section 2) for dividing a given population into a number of (say h) strata, such that each stratum has significantly lower internal variance. If a density distribution of the population is available (say $f(y)$), this algorithm computes the cumulative of the function $\sqrt{f(y)}$ and chooses the strata boundaries so that they make equal intervals on this cumulative scale.

We use this algorithm to divide each fundamental region into h strata, thus generating a total of $h*r$ *finer* fundamental regions, which become our strata. In our implementation, we build a histogram for each fundamental region, which approximates the density distribution, after which the stratification into h strata is easily accomplished. This can be done by a single scan of R. We use an equi-width histogram in the above step, although other kinds of histograms are also possible. A further refinement of this approach would be to divide each fundamental region R_j into h_j regions where h_j depends on the variance of R_j – the greater the variance, the larger the h_j. However, for simplicity we set the same value h for all regions (h set to 6 as suggested in Cochran [1977]), which gave good results in our experiments.

Let y_j be the average of the aggregate column values of all records in new stratum R_j. For the remainder of this section we make the following simplifying assumption. Since the variance of the values within any stratum R_j is small (due to stratification), we assume that each value within the stratum can be approximated as y_j. Thus R may be viewed as being partitioned into $h*r$ strata, where each stratum R_j has n_j records, each with the same aggregate column value y_j. Although clearly an approximation, such a view of R makes the subsequent mathematical analysis considerably simpler (and at the same time does not sacrifice much of the accuracy of the approximate query answering procedure).

4.4.2 Allocation

The structure of the allocation step is similar to COUNT, i.e., it is expressed as an optimization problem with $h*r$ unknowns $k_1 \ldots k_{h*r}$. However, there is a key difference. Unlike COUNT, here the specific values of the aggregate column y_j in each region influence $MSE(p_{\{Q\}})$. The following lemma shows how to express $MSE(p_{\{Q\}})$ as a function of the k_j's (the lemma assumes that the aggregate column values of R are either all positive or all negative).

Lemma 8: *For a SUM query Q in **W**, let*

$$ApproxMSE(p_{\{Q\}}) = \frac{\sum_{R_j \subseteq R_Q} \frac{n_j^2}{k_j} y_j^2 \delta(1-\delta) + \sum_{R_j \subseteq R \setminus R_Q} \frac{n_j^2}{k_j} y_j^2 \gamma(1-\gamma)}{\left(\sum_{R_j \subseteq R_Q} \delta n_j y_j + \sum_{R_j \subseteq R \setminus R_Q} \gamma n_j y_j\right)^2}$$

Then $\lim_{n\to\infty} \frac{MSE(p_{\{Q\}})}{ApproxMSE(p_{\{Q\}})} = 1$

The proof has been deferred to the appendix. As with COUNT, we can define $ApproxMSE(p_W)$ for SUM which is functionally of the form $\sum_{1\le j\le r} \alpha_j / k_j$, although the exact value of α_j is different from COUNT (each α_j depends on the same parameters n_1, $\ldots n_{h*r}$, δ, and γ, and additionally on the numbers $y_1, y_2, \ldots, y_{h*r}$). We can therefore use the same procedure for minimization as in Lemma 7.

4.5 Implementation Issues: Identifying Fundamental Regions

During the offline process of building a sample, we use a technique that we refer to as *tagging* to identify fundamental regions in relation R for a workload **W** consisting of selection queries. Tagging (logically) associates with each record $t \in$ R an additional column called *TagColumn* (of type varchar) that contains the list of queries in **W** that reference *t*. In our implementation, rather than adding *TagColumn* to R, we separate this column out into a different relation R' for two reasons. First, from a pragmatic standpoint, users do not want to change the schema of their database if avoidable. Second, we found that it is significantly faster (3X-5X in our experiments) to update the *TagColumn* in a separate relation R'. Records in R' have a one-to-one correspondence with records in R. This is done by including the key column(s) of R in R'. When a query Q $\in$ W is executed, for each record in R required to answer Q, we append the query-id of

Q to *TagColumn* of the corresponding record in R'. When R' is sorted by *TagColumn*, records belonging to the same fundamental region appear together. We experimentally evaluate the overhead of tagging in Section 8. We note that techniques reported in Ganti et al [2000] can be used to further reduce the cost of tagging records. Also, for selection only queries, we can also use a bit vector representation for *TagColumn* (instead of varchar) where the number of bits is equal to the number of queries in the workload. In this representation, bit *i* is set if query Q_i requires this record to answer the query. However, this representation is not possible for queries with GROUP BY since the tag also needs to encode the group. Finally, the following efficient method that requires only a single scan of R is also possible for single-table queries. For each record we tag it with all queries in the workload that select it. We can check if a query Q selects the record by applying the conditions in the WHERE clause in Q.

4.6 Putting it All Together

Figure 4 summarizes the key steps in STRAT and analyzes their complexity. The tagging step (Step 1) is I/O bound and dominates the running time of STRAT in practice (see Section 8); its running time is dependent on the number of queries in the workload. Steps 2-3 identify the fundamental regions in the relation for the given workload W and can be accomplished in time O($n$log$n$) where n is the size of the relation. Thus, Steps 1-3 constitute the *stratification* step of STRAT. Steps 4-5 constitute the *allocation* step (which is CPU bound) and runs in time O($q.h.u$), where q is the number of queries in **W**, u is the number of fundamental regions. Finally, Step 6 is the *sampling* step that actually generates the sample(s) from the source relations, and can be done in once scan of each source relation.

1. For each query Q $\in$ **W,** tag records in R used to answer query Q.
2. Identify the fundamental regions $R_{1\ldots}$ R_r.
3. For SUM queries, further divide each fundamental region R_j into *h* finer regions.
4. For each query Q $\in$ **W**, compute α_j of each (finer) region R_j referenced in Q, according to the formulas in Lemmas 6 and 8. At the end of this step, we have computed an α_j for each R_j.
5. Solve the optimization problem of distributing *k* records to regions. Let k_j be the number of records allocated to region R_j.
6. Perform stratified sampling to pick k_j records from region R_j and generate a sample of R.

Figure 4. Algorithm STRAT

5. EXTENSIONS FOR MORE GENERAL WORKLOADS

In this section, we consider further extensions to STRAT necessary to handle more general types of queries that may appear in the workload.

5.1 Extensions for a Heterogeneous Mix of Queries

Let us consider a workload that contains a mix of COUNT and SUM(y) queries. The lifting model (see Section 3) can be extended for such workloads easily. We need to make sure that each term $MSE(p_{\{Q\}})$ is set up appropriately to reflect the type of query Q in the workload since, as explained above, analysis for COUNT and SUM differ. Once these expressions are set up, minimizing the resulting $MSE(p_{\mathbf{W}})$ is straightforward.

Now consider a mix of queries such as SUM(x), SUM(y), SUM(x*y+z), etc. (where x, y, z are different columns from the same relation). We cannot directly apply the technique described in Section 4.4.1 for further stratifying each fundamental region to reduce variance because that technique works only for one-dimensional data. That is, if we stratify with the objective of reducing variance of x, the resulting stratification may not reduce variance of y. What we need is a solution to the *multivariate stratified sampling* problem (see Section 2). Although several works in the statistical literature have developed algorithms for this problem, for our purposes we found that the following simple (yet efficient) heuristic was adequate.

First we treat each expression (such as x*y+z) that appears in a query in the workload as a new (derived) column x'. Thus our workload is reduced to queries of the form SUM(x) where x is either one of the original columns or a derived column. Let X = {x_1, x_2, …, $x_{m'}$} be the set of (original or derived) columns that appear in the workload. We associate a weight w_i with each column x_i (where w_i is the sum of weights of all queries that aggregate on x_i). Essentially, the weight defines the *importance* of the column.

Next, we stratify each fundamental region into h strata by using a simple variant of the *h-means* clustering algorithm (Mitchell [1997]). This algorithm takes as inputs n multi-dimensional points and a parameter h, and produces h clusters such that the sum of the squared distances of each point from its cluster center is minimized (actually the h-means algorithm only produces a local optima). In our application we treat each record as an m'-dimensional point, and the output clusters are the strata. Our implementation differs from the classical h-means algorithm in the following ways. We use a "skewed" notion of distance, i.e., the (squared) distance between two points $t_1 = (x_{1,1}, x_{1,2}, \ldots, x_{1,m'})$ and $t_2 = (x_{2,1}, x_{2,2}, \ldots, x_{2,m'})$ is defined as $\sum_{1 \le i \le m'} w_i \left(x_{1,i} - x_{2,i}\right)^2$.

Thus the more important dimensions play a more dominant role in the distance. The intuition is that by minimizing the sum of such squared distances between points and their cluster centers, we will be able to significantly reduce the sum of the variance along all dimensions for all strata. Since the number of dimensions *m'* can become potentially large, we adopt a simple dimensionality reduction technique, where we discard all but a few of the most important columns. As part of our ongoing work, we are exploring other alternative dimensionality reduction techniques that take into account the correlations between the columns.

The other complication is that the *h*-means algorithm may perform a large number of iterations over the dataset before it converges to a local optimum. We avoid this problem by first selecting a small uniform sample of the records in each fundamental region, then running the *h*-means algorithm for a small constant number of iterations on this sample, and finally assigning each record of the fundamental region to its closest cluster center found in step (b).

We can also extend our techniques described in Section 4 to handle cases when the workload consists of single-table selection queries with aggregation but where each query can potentially reference a *different* relation. Although it may appear that we first need to partition the available memory for the sample across the tables and then pick samples from each table, our techniques are general enough to solve this problem in one step (as in Section 4 for single-table queries). In other words, once the relations have been tagged to get the entire set of fundamental regions across all relations, we can set up $MSE(p_{\mathbf{W}})$ similar to the single-table case and minimize it. The fact that the space requirement for each record of different relations is different must be taken into consideration in the allocation step.

5.2 GROUP BY Queries

We first show how workloads containing GROUP BY queries can be lifted (see Section 3 for how a workload containing pure selection queries can be lifted). Consider a GROUP BY query Q with weight *w* in the workload. Let Q partition R into *g* groups: $G_1 \ldots G_g$. Within each group G_j, let S_j be the set of records selected. We adopt the following simple lifting model: replace Q in the workload with *g* separate selection queries (each of weight *w*/*g*) that select $S_1, \ldots, S_g$ respectively, and use the techniques in Section 3 for lifting the resultant workload. Once we know how to lift a workload containing GROUP BY queries, adapting our algorithm for handling such a workload is straightforward. The tagging step treats each GROUP BY query Q as a collection of *g* selection queries with aggregation, and tags the records with the group that they belong to. During the tagging process, for GROUP BY columns of integer data types, we append a double <*c*, *v*> in addition to the query-id, where *c* is the column id of the GROUP BY column and *v* is the value of that column in record *t*. For non-integer data types, we treat the value of the GROUP BY column as a string and use a string hashing function to generate an integer value. As described in Section 4.5*,* when R' is sorted on *TagColumn*, all records belonging to the same fundamental region appear together.

5.3 JOIN Queries

Our algorithm can be naturally extended to a broad class of queries involving foreign key joins over multiple relations. Let us say that a relation is a *fact* relation in the schema if it references (i.e., contains the foreign keys of) one or more reference relations but is not referenced by any other relation. A relation is a *dimension* relation if it is does not contain foreign keys of any other relation. Thus, a relation that is neither a fact relation nor a dimension relation must be referenced by one or more relations and must contain foreign keys of one or more relations. We define *star query* to be a query that (a) is over the join of exactly one source relation and a set of dimension relations each of which is referenced by the source relation (b) has Group By and aggregation over a column of the source relation and (c) may have selections on source and or dimension relations. Star queries are widely used in the context of the decision support queries. In this section, we discuss how the technique that we have proposed in this paper can be extended for star queries in a straightforward manner. Our approach to handling star queries is as follows. We intend to obtain a sample only over the source relation. When a query is posed, we can then use the sample over the source relation to join the dimension relations in their entirety with the sample to compute the aggregate (with appropriate use of *ScaleFactor*). This method is reasonable because typically the source relation is a large fact relation (where sampling helps), while the other relations are smaller dimension relations.

Let us now consider how to pick a sample over the source relation. First, note that our model for lifting (see Section 3) will be based on subsets of the source relations selected in the workload, much like selection queries with aggregation. However, note that a record *t* in the source relation is deemed useful for a query Q in the workload only if *t* contributes to at least one answer record of Q, i.e., *t* must successfully join with other dimension relations and satisfy all the selection conditions in the query as well. For example, consider a query Q: "SELECT Sum(Sales) FROM *Sales* S, *Product* P WHERE S.*ProductId* = P.*ProductId* AND P.*ProductType* = 'Clothes' AND S.*Month* = 'Jan'". In this query *Sales* is the fact relation and *Product* is the dimension relation. During the tagging step of *Sales* for query Q, we only tag the records from *Sales* that join with *Product* and satisfy *both* the selection conditions in Q. The tagging step itself is no different from the technique used for single relation queries described in Section 4.5. For a workload that consists of star queries over multiple source relations, the technique described in Section 5.1 for selection queries over multiple relations is used.

We note that an alternative approach is to compute *join synopses* as in Acharya et al [1999], which results in reduced run-time cost at the expense of increased storage requirements due to additional columns from the join. Once again, allocation of k between the different synopses can be done by setting up $MSE(p_{\mathbf{W}})$ and minimizing it.

We conclude this discussion by pointing out the types of join queries where our approach either will not work, or needs to be investigated further before it can be made to work. Our approach will not work for joins involving two large relations that each need to be sampled, even if the join is a foreign key join. This is because it is known that joining the uniform random samples of two relations does not result in a uniform random sample of the joined relation, even for foreign-key joins (see Chaudhuri et al [1999]).

5.4 Extensions for Other Aggregates

In principle, a sample created using any algorithm (including STRAT) can be used to answer a query containing any aggregate function. However, since samples chosen by STRAT are optimized for workloads involving COUNT and SUM queries, the errors may be more for queries that involve other aggregates. Observe that a query Q involving AVG(y) can be estimated at run-time as SUM(y)/COUNT. Optimizing for workloads that contain AVG queries is therefore more difficult since $MSE(p_{\{Q\}})$ is more complicated to compute. In view of this difficulty, in our implementation we used a simple heuristic of treating an AVG query (with weight w) as a pair of SUM and COUNT queries (each with weight $w/2$).

6. SAMPLE MAINTENANCE

Approximate query processing is most relevant for decision support applications running on a data warehouse - where, unlike an OLTP environment, the collected data is mainly historical and consequently relatively static. Nevertheless, preprocessing a stratified sample from scratch every time the database has changed significantly can pose unacceptable overheads. In this section we discuss techniques for efficiently maintaining the stratified sample in the presence of updates to the database. Two types of updates are of concern to us: (a) insert/delete of data records, and (a) insert/delete of queries in the workload. Our focus is mainly on the first type of update.

6.1 Insert/Deletes of Data Records

Recall that the process of producing a stratified sample of a table R "from scratch" involves two tasks: (a) the process of tagging the database to discover the fundamental regions, and (b) the process of allocating the samples from the respective regions. Assume that at a previous point in time, the database table R has already been processed and a stratified sample created, but since then a set of new records R' have been collected (|R'| << |R|) that need to be inserted into R. We discuss deletions of data records later. We also assume that the workload is unchanged – handling a changing workload is discussed in the next subsection.

The basic idea is to continuously maintain, for each fundamental region, a reasonably large uniform random sample of fixed size (say K_j), called the *backing sample* of the region. Maintaining such a backing sample of a fixed size under the presence of data updates can be done using the techniques of Gibbons et al [1997] based on reservoir sampling techniques (Fan et al [1962]; Vitter [1985]), as well as techniques developed in Jermaine et al [2004] in case the backing sample is too large to be memory resident. Essentially, we simply have to tag the new records in R' to determine the regions to which they belong, and update each affected region's backing sample.

Next, the new allocations of the stratified samples from each region have to be determined. Note that this is necessary since the number of records n_j of some of fundamental regions have increased due to the addition of the new data, thus affecting the allocation formulas. Let k_j be the number of new allocated samples for region R_j. The new k_j may be either higher or lower than the old k_j, but the hope is that it is always smaller than K_j. Once the new allocations have been determined, the next task is to draw k_j random records from the backing sample of R_j. If the backing sample is kept in random order, this can be efficiently done by simply copying the first k_j records. And in the rare case that the backing sample is smaller than the allocated samples (i.e., $k_j > K_j$), we copy the entire backing sample to the stratified sample – thereby compromising the optimality of the original STRAT approach.

To avoid this latter situation, each K_j should be selected with some care. A simple strategy (that yielded good results in our experiments) is to let each K_j be a fixed generous fraction of the original size of the fundamental region (e.g., 5% or 10%).

Another strategy is to perform the first few re-allocations from scratch, and then assign each K_j to be a small multiple (e.g., twice) the largest k_j found thus far.

To analyze our approach, notice that the cost of tagging is purely incremental, as we only have to tag the new records that are inserted into the database. As mentioned earlier, the cost of maintaining the backing sample table per fundamental region can be efficiently done using the technologies developed in Gibbons et al [1997] and Jermaine et al [2004]. The cost of computing the reallocations is dependent on the number of fundamental regions and on the queries of the workload, but is independent of the database size. Finally, drawing k_j samples from each backing sample is a trivial operation. Thus the total time taken is dependent on R' and the total size of all backing samples, but is independent of the database size, n. For example, if each backing sample K_j is approximately 5% of each n_j, then we have approximately a twenty-fold speedup in the cost of updating the stratified sample when compared to computing the stratified sample from scratch.

We note that the backing sample solution can handle deletes, as backing samples can be maintained in the presence of deletes (Gibbons et al [1997]). Finally, we note that unlike computing stratified samples from scratch, in our incremental solution the backing samples have to be maintained in addition to the stratified sample, leading to extra disk usage. However, since disk space is a (relatively) cheap resource, this situation is not unduly disadvantageous.

6.2 Insert/Deletes of Queries in Workload

Although the stratified sample has been designed to be resilient to variations in the query workload (this is the main idea behind the "lifting" concept discussed in Section 3), in the case when the workload has significantly changed – e.g., very different analysis and reporting queries are introduced into the workload - the stratified sample needs to be updated. Detecting such significant changes in query patterns is usually the task of the database administrator, although as future work we are looking into ways in which such changes can be detected automatically – e.g., if we see a sudden increase in the errors of the approximations when compared to that of uniform random sampling. In such case, we can drop the current stratified sample and reconstruct a fresh stratified sample from scratch using the new workload.

7. RELATED WORK

Approximate query processing (AQP) for decision support in relational databases has been the subject of extensive recent research; see tutorials by Das [2003] and Garofalakis et al [2001] on this topic. Most research has focused on AQP systems that make use of concise data structures called *synopses* built from the database. The synopses techniques can be divided into two broad categories: non-sampling based, and sampling based. Sophisticated non-sampling based synopses such as wavelets (Chakrabarti [2000]; Vitter and Wang [1999]; Vitter et al [1998]), histograms (Ioannidis and Poosala [1999]; Poosala and Ganti [1999]), kernels (Gunopulos et al [2000]), probabilistic graph models (Getoor et al [2001]), and models for large sparse binary data (Pavlov et al [2003]) have been proposed as useful tools for AQP. In fact, histograms have a long tradition of use in selectivity estimation within query optimizers. However, these approaches are mainly suitable for all-numeric datasets. Furthermore, they seem to be limited to work for queries with restricted selection conditions, typically a conjunction of range conditions over individual attributes. As noted in Vitter and Wang [1999], a general problem with histogram-based approaches is that they incur high storage overhead and construction cost as the dimensionality of the data increases. In Vitter and Wang [1999] and Vitter [1998], the authors argued for the effectiveness of wavelets for handling aggregations over (high-dimensional) OLAP cubes. More recently, Chakrabarti et al [2002] showed how SQL operators can be applied directly on wavelet coefficients to efficiently produce approximate answers. However, the main disadvantage of all these non-sampling based approaches is that, while they are of great theoretical interest, their practical impact is limited due to the extensive modifications necessary to query processors and query optimizers to make use of these technologies.

Partly for the above reason, sampling-based systems have in recent years been the most heavily studied type of AQP system. Sampling-based systems have the advantage that they can be implemented as a thin layer of middleware which re-writes queries to run against sample tables which themselves can be viewed as ordinary relations in a standard, off-the-shelf database server. Thus, with sampling-based approaches, changes required to the database server are minimal. Secondly, the class of queries that can be solved by sampling-based approaches is much larger, e.g., *arbitrary* selection conditions are allowed in the queries.

Sampling-based approaches may themselves be categorized into two broad categories: on-the-fly samples and pre-computed samples. Hellerstein et al. [1997] describe *online aggregation* techniques that are of the latter category, in which approximate answers for queries are produced during early stages of query processing and gradually refined until all the data has been processed. The online aggregation approach has some compelling advantages: it does not require pre-processing of the data; it allows progressive refinement of approximate answers until the user is satisfied or the exact answer is supplied, and it can provide confidence intervals that indicate the uncertainty present in the answer. However, there are two important systems considerations that represent practical obstacles to the integration of online aggregation into conventional database systems. First, stored relations are frequently clustered by some attribute, so accessing tuples in a random order as required for online aggregation requires (slow) random disk accesses. Second, online aggregation necessitates significant changes to the query processor and user interface of a database system.

Due to the difficulty of purely online approaches to AQP, most research has focused on systems that make use of samples built by pre-processing the database. The AQUA project at Bell Labs (Acharya et al [1999]; Acharya et al [2000]) developed a pre-computed sampling-based system for approximate query answering. Techniques used in AQUA included *Join Synopses* (Acharya et al [2000]), which allow approximate answers to be provided for certain types of join queries, and *Congressional Sampling* (Acharya et al [1999]), which produces a stratified sample intended to minimize approximation error over a large set of GROUP BY queries. The problem of sampling-based approximate answers for join queries was also addressed in Chaudhuri et al [1999], which includes several strong negative results showing that many join queries are infeasible to approximate using sampling-based approaches.

Besides congressional sampling, several other non-uniform sampling techniques have been proposed that outperform uniform random sampling for certain types of queries. Workload information was used in Ganti et al [2000] to construct "self-tuning" *weighted samples* that adapted to the query workload. Chaudhuri et al [2001] proposes a technique called *Outlier Indexing* for improving sampling-based approximations for aggregate queries when the attribute being aggregated has a skewed distribution. In Jermaine [2003], an approach called *Approximate Pre-Aggregation* is presented, which is based on combining pre-computed samples with a small set of statistics of the data to improve accuracy, mainly for SUM queries, over data with large numeric variance. However, this approach seems to only work for queries with specific types of selection conditions, similar to histogram-based approaches. The technique of *Dynamic Sampling* is proposed in Babcock et al [2003], which attempts to strive a middle ground between on-line samples and pre-computed samples – a large family of differently weighted random samples are computed during the pre-processing phase, and then for each query that arrives during the runtime phase, an appropriate small subset of the samples is selected that can be combined to give a accurate approximate answer to the query.

Some of the most studied methods for addressing the long running times of data analysis queries are not approximation methods. Rather, techniques such as OLAP query processing and materialized view techniques have been designed to more efficiently produce *exact answers* to analysis queries. Examples of this class of techniques include constructing views of "data cubes" (Gray et al [1997]) and building indexes targeted at analysis queries (Agrawal et al [2000]; Chaudhuri et al [2000]; Chaudhuri and Narasayya [1998]). View matching and query containment have been well studied; see (Goldstein and Larson [2001]; Chaudhuri et al [1995]; Halevy [2001]) and the references therein. Likewise, the task of pre-computing an optimal set of materialization of views in physical database design is a widely studied problem in recent years (Agrawal et al [2000]; Agrawal et al [2004]). These physical data design techniques typically make use of significant pre-processing time and space, but they are often quite effective at speeding up specific queries. However, since it is prohibitively expensive to build indexes or materialized views sufficient to cover all possible queries, such techniques are of limited value for answering ad-hoc analysis queries, or even queries that bear resemblance to but are not the same as a previously encountered query; inevitably there will be certain unanticipated queries that "fall through the cracks" and are not aided by physical design, particularly in exploratory data mining and decision support applications.

Finally, we note that approximate query processing problems have been studied in areas beyond aggregation queries for decision support in relational databases. For example, AQP problems have cropped up in other domains such as in data stream processing (e.g., see Koudas and Srivastava [2003]; Garofalakis [2005]; Greenwald and Khanna [2001]). However, the emphasis in these applications is quite different from our application of analytical querying of a data warehouse – the focus is on maintaining aggregates for fixed queries and streaming data using space-efficient techniques. AQP problems have also been considered in sensor networks (Kempe et al [2003]), and in XML databases (Polyzotis et al [2004]). However, a detailed discussion of these efforts is beyond the scope of this paper.

In the rest of this section, we focus in more detail on three sampling based approaches, congressional sampling (Acharya et al [2000]), outlier indexing (Chaudhuri et al [2001]) and weighted sampling (Ganti et al [2000]), since all three are purely based on pre-computed samples and hence most closely related to our approach. We make a careful comparison of these techniques against our own approach, from both theoretical as well as conceptual points of view. In Section 8, we also provide an extensive experimental comparison of these methods with our solutions.

7.1 Comparison against Weighted Sampling

The approach in Ganti et al [2000] (and also a part of Chaudhuri et al [2001]) is based on a seemingly intuitive idea of weighted sampling. Each record in the relation R to be sampled is tagged with a frequency, i.e., the number of queries in the workload such that the record must be selected to answer the query. Once the tagging is done, an expected number of *k* records are selected in the sample, where the probability of selecting a record *t* with frequency f_t is $k\left(f_t \Big/ \sum_u f_u\right)$. Thus, records that are accessed more frequently have a greater chance of being included inside the sample. However, this method preferentially allocates more records to "larger" queries, i.e., queries with higher selectivity. To see this, consider a set of *q* queries $\{Q_1, \ldots Q_q\}$ in the workload such that a few queries reference large portions and most queries reference very small portions of the database. Then, by the weighted sampling scheme described above most records in the sample will come from the large portions. Therefore, the approximate answers for the larger queries will be much more accurate than for the smaller queries. A better strategy is to pick the same number of records for each query and thereby answer all queries with the same accuracy. Another shortcoming of the weighted sampling approach (and in fact, of all approaches prior to our paper) is that it assumes a fixed workload, i.e., the technique does not cope with uncertainty in the expected workload. Furthermore, Ganti et al [2000] do not address the issue of variance of data in the aggregate column. However, a novelty of the paper is that it tackles the issue of maintaining and continuously refreshing a sample of records of R after a new query has been processed.

7.2 Comparison against Outlier Indexing

Chaudhuri et at [2001] attempted to address the problem of internal variance of data in the aggregate column. The basic idea is that *outliers* of the data (i.e. the records that contribute to high variance in the aggregate column) are collected into a separate index, while the remaining data is sampled using a weighted sampling technique. Queries are answered by running them against both the outlier index as well as the weighted sample, and an estimated answer is composed out of both results. This method too is easily seen to result in a sub-optimal solution, since the concept of an outlier index + a (weighted) sample can be viewed as a special type of our approach using stratified sampling, where the *outliers form their own stratum that is sampled in its entirety*. Moreover, the approach is only designed for fixed workloads and does not take into account uncertainties in the expected workload. The idea of separately handling outliers has also appeared in the context of applying exploratory data analysis methods on data cubes (Barbará and Sullivan [1997]; Barbará and Wu. [1999]).

7.3 Comparison against Congressional Sampling

The *congressional sampling* paper (Acharya et al [2000]) has the most principled approach of the three papers. The authors advocate a stratified sampling strategy called Congress that tries to simultaneously satisfy a set of GROUP BY queries. Some key concepts of our paper (e.g. the concept of fundamental regions) have been influenced by it. However, their approach is still ad-hoc in the sense that even though they try to reduce the error, their scheme does not minimize the error for any of the well-known error metrics. We illustrate this through the following example.

Consider two GROUP BY-COUNT queries Q_1 and Q_2 over a relation. Let Q_1 define only one group, g_1 (i.e. the entire relation R), while Q_2 defines two groups, a large group g_{21} with n_{21} records (where $n_{21} \approx n$) records and a very small group g_{22} with the remaining $n_{22} = n - n_{21}$ records. As Acharya et al [2000] assume, let the expected query distribution be such that each GROUP BY query is equally likely (but the selection conditions may vary, under the assumption that for each query the per-group selectivity is the same for all groups). Let k ($k > 0$) be the number of records to be selected in the sample. Assume a large population, i.e. that even n_{22} is large compared to k. Congress divides R into two strata R_1 and R_2 (essentially identical to g_{21} and g_{22} respectively). It allocates k samples between the two as follows:

$$k_1 = k\left(\frac{\max\left(k,\frac{k}{2}\right)}{\max\left(k,\frac{k}{2}\right)+\max\left(0,\frac{k}{2}\right)}\right) = k\left(\frac{2}{3}\right), k_2 = k\left(\frac{\max\left(0,\frac{k}{2}\right)}{\max\left(k,\frac{k}{2}\right)+\max\left(0,\frac{k}{2}\right)}\right) = k\left(\frac{1}{3}\right)$$

While this allocation has seemingly intuitive appeal, as we now show it does not minimize any of the well-known error metrics for the expected query distribution (such as *MSE* or L_1).

7.3.1 Minimizing MSE

For example, suppose we wanted to minimize the *MSE* of the expected query distribution. Let k_1 and k_2 be the (unknown) allocation of the samples in the two strata R_1 and R_2. The *MSE* of queries like Q_1 is proportional to $1/k_1$. The *MSE* of queries like Q_2 is proportional to $\frac{1}{2}\left(\frac{1}{k_1}+\frac{1}{k_2}\right)$. Thus the overall *MSE* is proportional to $\frac{1}{2}\left(\frac{1}{k_1}+\frac{1}{2}\left(\frac{1}{k_1}+\frac{1}{k_2}\right)\right)$, which is equal to $\frac{3}{4k_1}+\frac{1}{4k_2}$. Using simple differentiation techniques (as in the proof of Lemma 4), the overall *MSE* is minimized if $k_1 = k\left(\frac{\sqrt{3}}{\sqrt{3}+1}\right), k_2 = k\left(\frac{1}{\sqrt{3}+1}\right)$. Clearly these values are not the same as that allocated by Congress.

7.3.2 Minimizing L_1

Instead of *MSE*, suppose we wanted to minimize the L_1 error of the expected query distribution. The L_1 error of queries like Q_1 is proportional to $\frac{1}{\sqrt{k_1}}$. The L_1 error of queries like Q_2 is proportional to $\frac{1}{2}\left(\frac{1}{\sqrt{k_1}}+\frac{1}{\sqrt{k_2}}\right)$. Thus the overall L_1 error is proportional to $\frac{1}{2}\left(\frac{1}{\sqrt{k_1}}+\frac{1}{2}\left(\frac{1}{\sqrt{k_1}}+\frac{1}{\sqrt{k_2}}\right)\right)=\frac{3}{4\sqrt{k_1}}+\frac{1}{4\sqrt{k_2}}$. Using simple differentiation techniques, the overall L_1 error is minimized if $k_1 = k\left(3^{\frac{2}{3}}\Big/\left(3^{\frac{2}{3}}+1\right)\right)$ and $k_2 = k\Big/3^{\frac{2}{3}}+1$. Once again, these values are not the same as that allocated by Congress.

In summary, the following table summarizes the main differences between STRAT and competing sampling-based approximate query answering systems:

AQP Technique	**Leverages Workload?**	**Models Lifted Workload?**	**Handles Data Variance?**	**Handles Group By?**	**Optimizes Error?**
Outlier Indexing	yes	no	yes	no	no
Weighted Sampling	yes	no	no	no	no
Congress	yes	no	no	yes	no
STRAT	yes	yes	yes	yes	yes

Table 3. STRAT versus other Sampling-Based Approaches

8. EXPERIMENTAL RESULTS

We have implemented STRAT on Microsoft SQL Server 2000 and conducted experiments to evaluate its effectiveness. We compared their quality and performance with the following previous work: (a) uniform random sampling (USAMP) (b) weighted sampling (WSAMP) [Ganti et al [2000], Chaudhuri et al [2001]], (c) outlier indexing combined with weighted sampling (OTLIDX) (Chaudhuri et al [2001]), and (d) congressional sampling (CONG) (Acharya et al [2000]). We also performed simulation experiments to verify the accuracy of formulas established in Lemmas 6 and 8. We describe the implementation of the previous work, our experimental setup, the results of the experiments, and draw conclusions.

8.1 Implementation of Previous Approaches

We briefly describe our implementation of the previous works (the key implementation aspects of STRAT have already been discussed earlier). For uniform sampling (USAMP), each record is accepted with probability equal to the sampling fraction. We generate a uniform random sample in one scan of the relation R using the *reservoir sampling* technique (Fan et al 1962; Vitter [1985]). For weighted sampling (WSAMP) (Ganti et al [2000]; Chaudhuri et al [2001]) the probability of accepting a record is proportional to the frequency with which the record is selected by queries in the workload. We calculate this frequency for each record using the tagging technique described in Section 4.5. The key difference is that rather than keeping track of the list of queries that select the record, we only need a single counter (an integer) for the *TagColumn* to keep track of the frequency. For the outlier-indexing method (OTLIDX), we implemented the technique described in Chaudhuri et al [2001]. The paper does not address the following issue: for a given sample size, how many records of the sample to allocate for the outlier index, and how many to the weighted sample? To give OTLIDX the best possible choice of alternative settings, we tried different strategies for partitioning the sample for different databases and workloads – 25% for outliers-75% for weighted sample, 50%-50% and 75%-25%.

We use the 50%-50% strategy since it performed well for most workloads. We also implemented the congress algorithm (CONG) described in Acharya et al [2000]. The algorithm takes as input a set G of GROUP BY columns and builds a sample for answering queries on any subsets of G (including ∅). For each subset of G, it determines the best allocation for each of the finest groups in the relation. The final allocation for a group is proportional to the maximum allocation for that group over all subsets of G. Since the algorithm for Congress that takes into account selections in the workload is not publicly available, in our experiments we only evaluate Congress for workloads consisting of pure GROUP BY queries (i.e., no selections).

8.2 Experimental Setup

8.2.1 Hardware/OS

All experiments were run on a machine having an x86 550 MHz processor with 256 MB RAM and an internal 18GB hard drive running Microsoft Windows 2000.

8.2.2 Databases

We used the popular TPC-R benchmark for our experiments (see TPC benchmark specifications at www.tpc.org). One of the requirements of the benchmark however, is that the data is generated from a *uniform distribution*. Since we were interested in comparing the alternatives across different data distributions, we used the publicly available program (see Chaudhuri and Narasayya) for generating TPC-R databases with differing data skew. For our experiments we generated 100MB as well as 1GB TPC-R databases, and varied the Zipfian parameter *z* over values 1, 1.5, 2, 2.5, and 3 (see Zipf [1949]). The fact table (*lineitem*) of the 100MB (resp. 1GB) database had 600,000 (resp. 6,000,000) rows. We report a few relevant characteristics of the data in the aggregation column used. The ratio of the maximum to the minimum value in the aggregation column was varied between approximately 9000 and 250000 for the different databases. (For example, for z = 2, for one of the 100MB database the aggregation column had an average value of 26857.2861, and standard deviation of 11958.6707). Also, there is no correlation between values in the aggregation column (picked from the Zipfian distribution) and their frequency in the data.

8.2.3 Workloads

We generated several workloads over the TPC-R schema using an automatic query generation program. The program has the following features that can be turned on: (i) aggregations on the fact table (*lineitem*), (ii) foreign-key joins between the fact table and a dimension table (*part* or *supplier*), (iii) grouping and (iv) selection. We experimented with three classes of workloads containing aggregation: (a) W-SEL (Selections, Foreign-Key Joins). (b) W-GB (Group By, Foreign-Key Joins) (c) W-SEL-GB (Selections, Group By, Foreign-Key Joins. Thus, e.g., W-SEL-GB-100 indicates a workload from the W-SEL-GB class containing 100 queries. The selection conditions were on the following columns: *l_shipdate*, *l_orderkey*, *l_tax*, *l_discount*, *p_partkey*, *p_size*, *p_retailprice*, *s_acctbal*, *s_suppkey*. To enable a comparison against Acharya et al [2000], we used the three grouping columns *l_shipdate*, *l_returnflag* and *l_linestatus*. The aggregate column was *l_extendedprice*, and the aggregation expressions used were COUNT and SUM. For each workload, we used the first half of the workload as the training set that was used to determine the sample, and the second half as the test set. We controlled the degree of similarity between the training and test set using the following two parameters: (a) the set of columns on which conditions are allowed in the training set and in the test set, and (b) for each column on which a selection is defined, control the range of the selection condition.

8.2.4 Parameters

We varied the following parameters in our experiments: (a) Skew of the data, z (b) The sampling fraction *f* was varied between: 0.1% - 10%. (c) Workload size was varied between 25 - 800 queries. All numbers reported are the average over multiple runs.

8.2.5 Error Metric

As with previous work, we mainly report the average relative error over all queries in the workload, i.e., L_1 metric. We have found in our experiments that similar trends also hold for the *RMSE* (L_2) error metric.

8.3 Results

8.3.1 Quality vs. Sampling Fraction

We compare the quality (errors) of the various techniques for the COUNT and SUM aggregates as the sampling fraction is varied keeping the workload (W-SEL-GB-100) and data skew (*z* = 2) fixed. We report results for 100MB databases – similar trends were also observed for 1GB databases. As we see from Figures 5 and 6, for the test set (for COUNT and SUM aggregates respectively), the errors for STRAT are relatively low even with as little as 1% sampling whereas errors with other methods (USAMP, WSAMP, OTLIDX) are significantly higher. The key point to note for the SUM aggregate is that STRAT is able to achieve better quality than OTLIDX by taking into account the variance in the data values in a more principled way. (Note that in Figures 5 and 6 we express the sample sizes both in absolute counts as well as fractions of the (600,000 records, 100MB) fact table. To avoid clutter, some of the later charts only have the sampling fraction displayed.)

Next, we compare the quality of the various alternatives for the *training set* itself. We see the effectiveness of our stratification algorithm from Figure 7 (for the COUNT aggregate) and Figure 8 (for the SUM aggregate), where STRAT gives errors close to 0 once the sample size exceeds the number of fundamental regions.

For comparisons with CONG, we consider workloads with only GROUP BY queries (i.e., no selection). Figure 9 (test set) shows that for the COUNT aggregate, STRAT performs best among all methods. We note that CONG also does significantly better than the other methods. The reason STRAT is more accurate than CONG is that despite attempting to account for all groups, CONG still allocates too many records to large groups and not enough for small groups, whereas STRAT is able to balance the allocations better. For GROUP-BY queries with the SUM aggregate, we see from Figure 10 (test set) that once again STRAT performs best among all methods. However, OTLIDX appears to perform better than CONG since unlike STRAT and OTLIDX, CONG does not take into account the data variance when allocating samples.

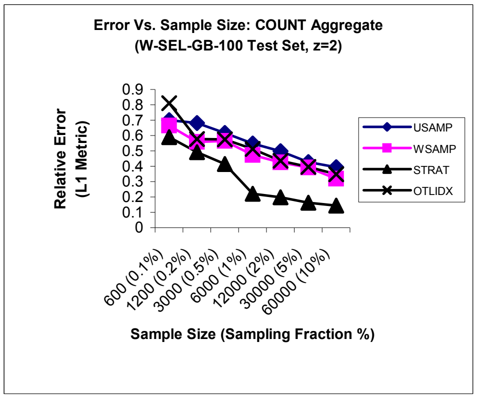


Figure 5. COUNT Aggregate – Test Set

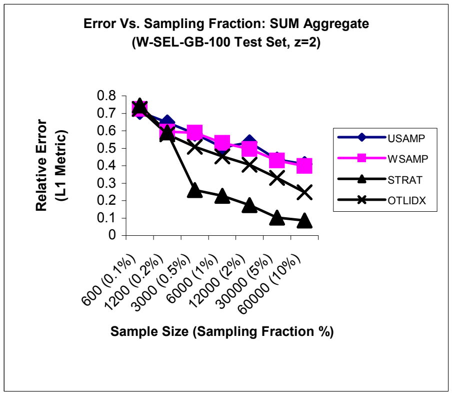


Figure 6. SUM Aggregate – Test Set

8.3.2 Quality vs. Overlap between Training Set and Test Set

We vary the degree of overlap of the minimum and maximum values of the range from which selection conditions are generated. The degree of overlap is an informal measure of *correlation*. For example, a degree of overlap of 0% (negative correlation) implies that for each column in a selection condition, the range of values from which selection conditions can be chosen for the test and training set for each column are disjoint, whereas 100% overlap (positive correlation) implies that the ranges are the same. From Figure 11 we see that for small overlap, as expected STRAT (δ = 0.90, γ = 0.01) gives higher errors than other methods. However, for moderate to large overlaps, STRAT is significantly better. All reported results are for 100MB databases – similar trends were also observed for 1GB databases.

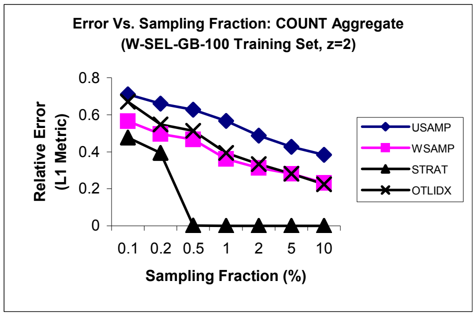


Figure 7. COUNT Aggregate – Training Set

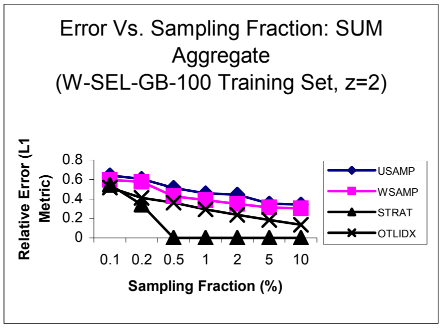


Figure 8. SUM Aggregate – Training Set

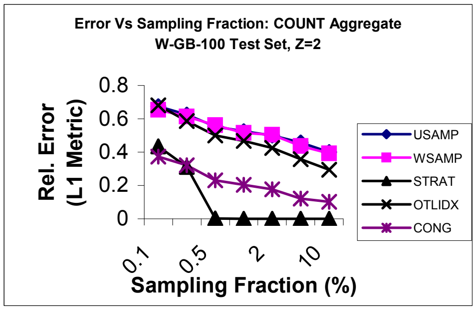


Figure 9. GROUP BY only workload. COUNT Aggregate – Test Set

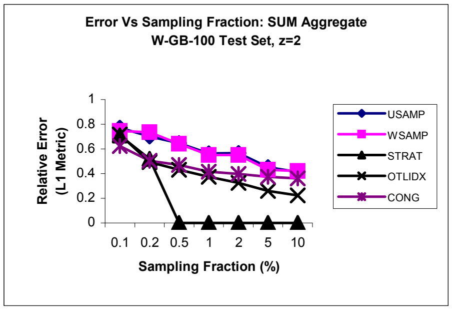


Figure 10. GROUP BY only workload. SUM Aggregate – Test Set

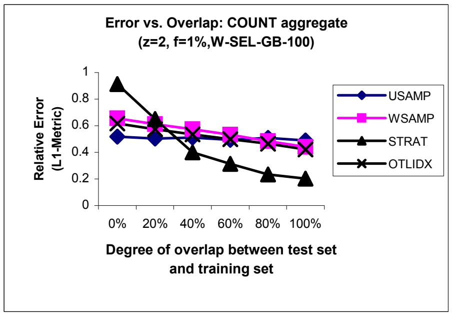


Figure 11. Varying overlap between training set and test set

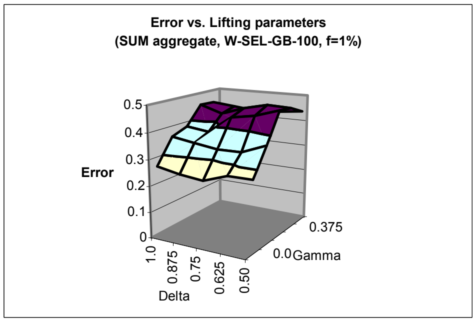


Figure 12. Error vs. lifting parameters for test set

8.3.3 Automatically Determining the Lifting Parameters δ and γ

For a given workload W-SEL-GB-100 and sampling fraction of 1% (for 100MB databases), Figure 12 shows how the error for the test set varies with δ and γ (see Section 4). We see that the error varies gradually, which indicates that our grid search approach is promising.

8.3.4 Quality vs. Data Skew

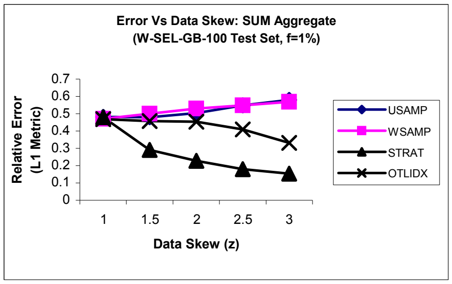


Figure 13. Variation in Data Skew: SUM Aggregate

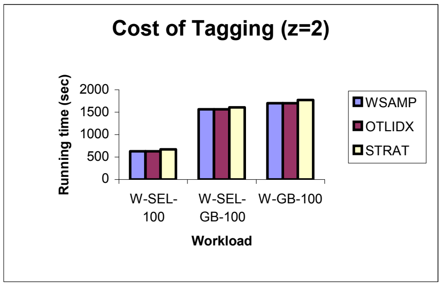


Figure 14. Comparison of running time to build sample

In this experiment we compared the quality of the different methods as the skew of the data (z) is varied between 1 and 3, keeping the workload (W-SEL-GB-100) and sampling fraction (1%) fixed, for the SUM aggregate. We find (see Figure 13) that for moderately skewed to highly skewed data (*z* > 1), STRAT gives significantly lower errors than other methods (by about 20%). For low skew data (*z* = 1), the other methods are comparable to STRAT.

8.3.5 Comparison of Time for Building Samples

We compare the time to build the sample for WSAMP, OTLIDX, and STRAT for three different workloads of 100 queries each. Figure 14 (for 100 MB database, data skew *z* = 2) shows that the additional time relative to WSAMP taken by STRAT to tag the database (Section 4.5) for the given workload is small. The difference between the tagging for WSAMP and STRAT is that in STRAT we additionally need to record the query-id information (and for GROUP BY queries, the group information). Finally, for a 1% sample, we report that the time to actually pick the sample after tagging was 15 sec, 70 sec, and 36 sec respectively for WSAMP, STRAT and OTLIDX for the W-SEL-GB-100 workload. Thus, the total time to build a sample is dominated by the time taken to tag the relation for the given workload.

Additional experiments on preprocessing of larger IGB databases are described in Section 8.3.8.

8.3.6 Comparison on a Real Data Set

We compare the quality of various approaches on a real data warehouse within our organization, used to track sales of products. We used δ = 0.90 and γ=0.01 for STRAT. We used a portion of the database of approximately 0.84 million rows; training and test sets of 25 real queries used by the application each. These queries typically contained 3-6 GROUP BY columns and 2-5 selection conditions per query. Figures 15 and 16 show (respectively for the test set and the training set), STRAT performs consistently better than other methods for this real data set.

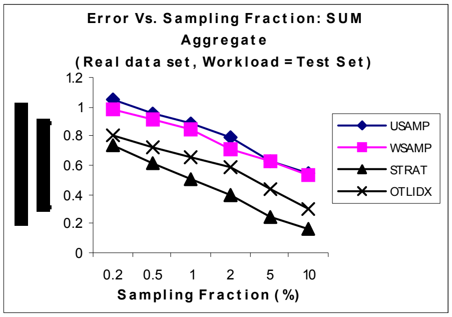


Figure 15. Error vs. Sampling Fraction (Test Set)

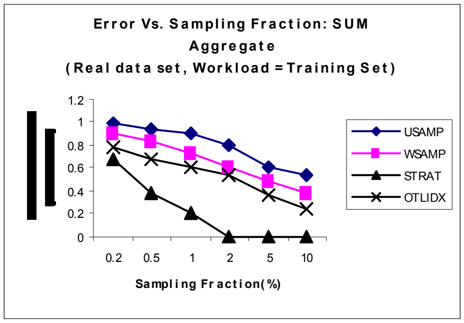


Figure 16. Error vs. Sampling Fraction (Training Set)

8.3.7 Results for L_2 Metric

We present results using the L_2 metric (*RMSE*) of our experiments on 100MB databases for the W-SEL-GB-100 workload for the test set for the COUNT aggregate (Figure 17) and the SUM aggregate (Figure 18). For corresponding numbers with the L_1 metric, please see Figures 6 and 7 respectively. As we see from the figures, the results have similar trends as those reported with the L_1 metric. The only difference is that the errors are relatively smaller (for all methods) using L_2.

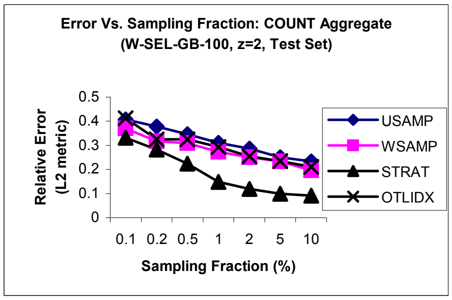


Figure 17. L_2 error for COUNT aggregate

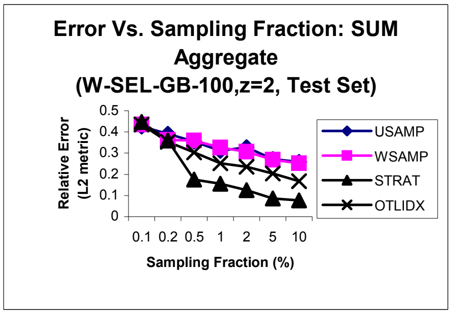


Figure 18. L_2 error for SUM aggregate

8.3.8 Results for Sample Maintenance Experiments

We carried out experiments to determine the effectiveness of our sample maintenance algorithms in the presence of data updates. Since the maintenance of a uniform random sample in the presence of data updates is a well-studied technique (see Gibbons et al [1997]; Jermaine et al [2004]), our focus in these experiments was to investigate the accuracy loss that resulted in preparing stratified samples from incrementally maintained backing samples of fundamental regions rather than preparing stratified samples from scratch.

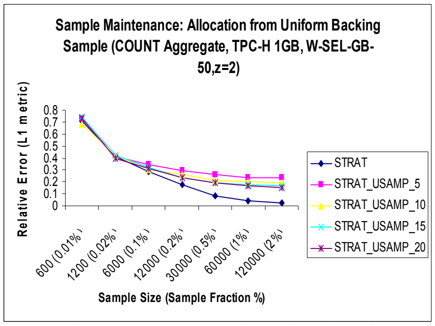


Figure 19. L_1 error when allocating from Uniform Backing Samples (COUNT)

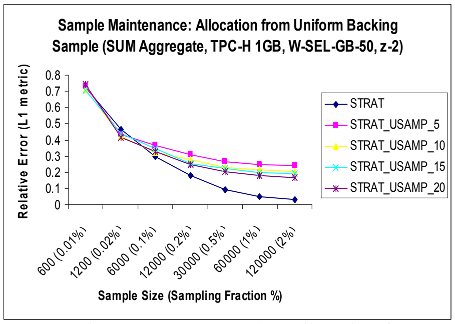


Figure 20. L_1 error when allocating from Uniform Backing Samples (SUM)

Our experiments tested a straightforward approach where the backing sample size was a fixed percentage of the size of each fundamental region. For these experiments we used a larger database than reported for the previous experiments - a 1GB TPC-H database (where the *lineitem* table contained 6,000,000 records) with a skew of z = 2, and the W-SEL-GB-50 workload. Figure 19 shows the loss in accuracy for COUNT queries when, instead of computing stratified samples from scratch, we draw the stratified sample from backing samples of size 5%, 10%, 15% and 20% respectively. As can be seen, for sampling fractions of 0.1% or less (which is reasonable for 1GB databases), the backing sample approaches provide accuracies that are comparable to the original STRAT algorithm. Figure 20 shows similar results for SUM queries.

Moreover, as Figure 21 shows, this comparable accuracy comes at a much faster maintenance cost – the cost of tagging and allocating the stratified sample from backing samples is much more efficient than re-computing the stratified sample from scratch. Investigating more sophisticated strategies – e.g., selecting the backing sample size K_j for each region more judiciously as described in Section 6 – is left for future work.

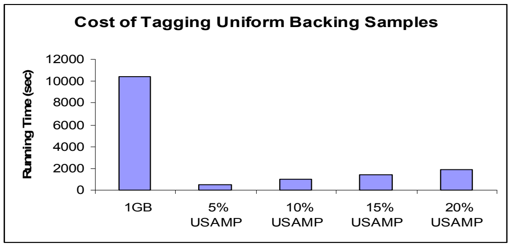


Figure 21. Cost of Tagging Uniform Backing Samples

8.3.9 Verifying Accuracy of Lemmas 6 and 8

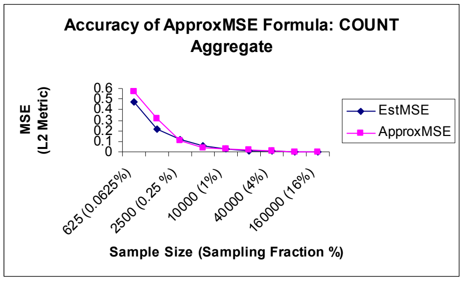
Although most of the above experiments indirectly establish that the formulas derived for ApproxMSE in Lemmas 6 and 8 are accurate (by demonstrating that the resultant errors using STRAT are smaller compared to competing approximate query processing schemes), we also ran separate simulation experiments to directly verify these formulas. We generated a table R with *n* records (i.e., *n* varied from one hundred thousand to ten million), and with a measure column that was generated according to a Zipfian distribution ($z = 2$). This table was arbitrarily partitioned into *r* strata (in our experiments, *r* was set to 100, and the sizes of the strata, n_j, were distributed according to a Zipfian distribution $z = 2$). A "master query" Q was designated to be a subset of the strata (the selectivity of Q was varied from 0.1% to 10%) - thus R_Q was the set of strata that belonged to the master query, and R \ R_Q was the set of remaining strata. Note that the query Q is simply defined as a set of strata (or equivalently, as a set of records), and not a SQL statement.

For this experiment, we tried different values of δ and γ, but report results for only a specific pair of values (0.9 and 0.05 respectively). We generated a "workload" $\mathbf{W_Q}$ of 100 queries, where each query Q_i in the workload was generated by scanning all records of R, tossing a coin with bias δ (resp. γ) if the record was inside (resp. outside) Q, and retaining the record if the coin came up heads.

Next, for each query Q_i in $\mathbf{W}_Q$, we prepared a stratified sample of R as follows. For each stratum (or fundamental region) R_j, we allocated a value k_j (for our experiments k_j was varied between 0.1% and 10%). Then, we drew k_j sample records from each stratum using reservoir sampling. Note that this stratified sample is not necessarily optimal – however this was not an issue since the main objective of this experiment was to verify that the formulas in Lemmas 6 and 8 are accurate.

We then "approximately answered" Q_i by joining its records with the sample records, and appropriately scaling and aggregating the records that survived the join. We then computed the relative squared error of answering this query, and estimated the mean squared error by averaging over the entire workload. This estimated mean squared error was compared against the approximate mean squared error derived in Lemmas 6 and 8.

Our experiments showed that the estimated mean squared error compared very well with the approximate mean squared error as derived in Lemma 6 for the COUNT aggregate. As Figure 22 shows (for a table with 1,000,000 rows and a master query with selectivity 1%), both errors decreased as the overall sampling fraction for the database increased, which was as expected. Similar behavior was also observed for the formula in Lemma 8, i.e., for the SUM aggregate. Moreover, the approximations improved as the overall sampling fraction increased. This can be seen in Figure 23, which shows (for the SUM aggregate) that the ratio of ApproxMSE/EstMSE approaches 1 with increasing sampling fraction.



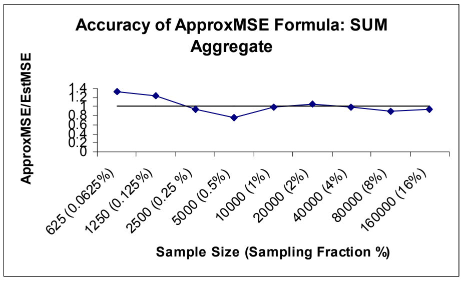


Figure 22. Approximate MSE versus Estimated MSE for COUNT aggregate

Figure 23. Approximate MSE versus Estimated MSE for SUM aggregate

We also observed that these approximations improved for larger database tables, when the overall sample size was kept a constant (i.e., when *n* is increased while *k* is kept a constant). Figure 24 shows how the approximation improves as the database table R gets larger, with *k* being fixed at 10,000 sample records (master query selectivity 1%).

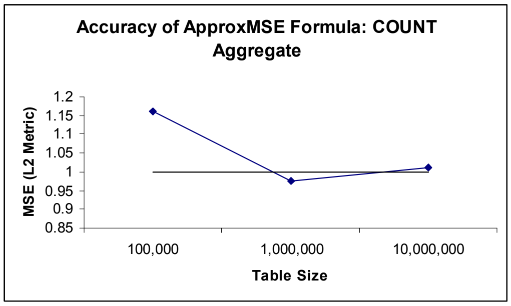


Figure 24. ApproxMSE versus Estimated MSE for COUNT aggregate as database size increases

9. CONCLUSIONS

In this paper, we present a comprehensive solution to the problem of picking pre-computed samples for approximately answering aggregate queries and show how it can be implemented in a commercial database system. Through a novel technique for lifting a workload, our solution can be tuned to work well even for workloads that are similar but not identical to the given workload. Our solution is robust since it also handles the problems of data variance, heterogeneous mixes of queries, GROUP BY and foreign-key joins.

As future work, we would like to generalize our algorithms to a more complete subset of SQL, such as for workloads containing joins other than foreign-key joins, as well as nested sub-queries. It would be also interesting to design approximate query answering systems that are a hybrid mix of sampling and non-sampling based approaches, especially since data statistics such as histograms are often available for free within a database system. Finally, dynamic sample selection (Babcock et al [2003]) is an exciting new area in approximate query processing that attempts to bridge the gap between the two extremes of online sampling and offline sampling, and it would be interesting to see whether principled approaches based on workload analysis can play a significant role in such efforts.

ACKNOWLEDGEMENT

We are grateful to Muhammed Z. Miah for helping with some of the experiments.

REFERENCES

ACHARYA S., GIBBONS P. B. AND POOSALA V. 2000. Congressional Samples for Approximate Answering of Group-By Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

ACHARYA S., GIBBONS P. B., POOSALA V. AND RAMASWAMY S. 1999. Join Synopses for Approximate Query Answering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.

AGRAWAL S., CHAUDHURI C. AND NARASAYYA V. 2000. Automated Selection of Materialized Views and Indexes in SQL Databases. In *Proceedings of the International Conference on Very Large Databases*, 2000, 496-505.

AGRAWAL S., NARASAYYA V. AND YANG B. 2004. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004, 359-370.

BABCOCK B., CHAUDHURI C. AND DAS G. 2003. Dynamic Sample Selection for Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, 539-550.

BARBARÁ D. AND SULLIVAN M. 1997. Quasi-Cubes: Exploiting Approximations in Multidimensional Databases. *SIGMOD Record*, 26(3).

BARBARÁ D. AND WU. X. 1999. Using Approximations to Scale Exploratory Data Analysis in Datacubes. In Proceedings *of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

BETHEL J. 1989. Sample Allocation in Multivariate Surveys. *Survey Methodology*, 15, 47-57.

CAUSEY B. D. 1983. Computational Aspects of Optimal Allocation in Multivariate Stratified Sampling. *SIAM Journal of Scientific and Statistical Computing*, 4 2.

CHAKRABARTI K., GAROFALAKIS M., RASTOGI R. AND SHIM K. 2000. Approximate Query Processing Using Wavelets. In *Proceedings of the International Conference on Very Large Databases*, 2000.

CHAUDHURI S., DAS G., DATAR M., MOTWANI R. AND NARASAYYA V. 2001. Overcoming Limitations of Sampling for Aggregation Queries. In *Proceedings of the IEEE International Conference on Data Engineering*, 2001.

CHAUDHURI S., DAS G. AND NARASAYYA V. 2001. A Robust, Optimization-Based Approach for Approximate Answering of Aggregation Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

CHAUDHURI S., R. KRISHNAMURTHY R., POTAMIANOS S. AND SHIM S. 1995. Optimizing Queries with Materialized Views. In *Proceedings of the IEEE International Conference on Data Engineering*, 1995.

CHAUDHURI S., MOTWANI R. AND NARASAYYA V. 1999. Random Sampling over Joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.

CHAUDHURI S. AND NARASAYYA V. Program for TPC-D Data Generation with Skew. http://research.microsoft.com/dmx/

CHAUDHURI S. AND NARASAYYA V. 1997. An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the International Conference on Very Large Databases*, 1997.

CHAUDHURI S. AND NARASAYYA V. 1998. AutoAdmin What-if Index Analysis Utility. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998.

CHROMY J. W. 1987. Design Optimization with Multiple Objectives. In *Proceedings of Survey Research Section*, American Statistical Association, 1987.

COCHRAN W.G. 1977. *Sampling Techniques*. John Wiley & Sons, New York, Third edition.

DAS G. 2003. Survey of Approximate Query Processing Techniques. *Tutorial at International Conference on Scientific and Statistical Database Management*, 2003.

FAN C. T. MULLER M. E. AND REZUCHA I. 1962. Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital Computers. *Journal of the American Statistical Association*, 57 298, 387-402.

GANTI V., LEE M. L. AND RAMAKRISHNAN R. 2000. ICICLES: Self-tuning Samples for Approximate Query Answering. In *Proceedings of the International Conference on Very Large Databases*, 2000.

GAROFALAKIS M. N. AND GIBBONS P. B. 2001. Approximate Query Processing: Taming the TeraBytes. *Tutorial at the International Conference on Very Large Databases*, 2001.

GAROFALAKIS M., GANGULY S. KUMAR A. AND RASTOGI R. 2005. Join-Distinct Aggregate Estimation over Update Streams. In *Proceedings of the International Conference on the Principles of Database Systems*, 2005.

GETOOR L., TASKAR B., AND KOLLER D. 2001. Selectivity Estimation using Probabilistic Model. In *Proceedings of the SIGMOD International Conference on Management of Data*, 2001.

GIBBONS P. B., MATIAS Y. AND POOSALA V. 1997. *Fast Incremental Maintenance of Approximate Histograms*. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

GOLDSTEIN J. AND LARSON P. 2001. Optimizing queries using materialized views: a practical, scalable solution. In *Proceedings of the SIGMOD International Conference on Management of Data*, 2001.

GRAY J., CHAUDHURI S., BOSWORTH A., LAYMAN A., REICHART D., VENKATRAO M., PELLOW F., AND PIRAHESH H.. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Journal of Data Mining and Knowledge Discovery*, 1(1), 29-53.

GREENWALD M. AND KHANNA S. 2001. Space-Efficient Online Computation of Quantile Summaries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

GUNOPULOS D., KOLLIOS G., TSOTRAS V. J. AND DOMENICONI C. 2000. Approximating Multi-Dimensional Aggregate Range Queries Over Real Attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

HALEVY A. Answering Queries Using Views: A Survey. 2001. *VLDB Journal*, 2001.

HELLERSTEIN J., HAAS P. AND WANG H. 1997. Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

HOCHBAUM D. 1997. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing.

IOANNIDIS Y. AND POOSALA V. 1999. Histogram Based Approximations of Set-Valued Query Answers. In *Proceedings of the International Conference on Very Large Databases*, 1999.

JERMAINE C. 2003. Robust Estimation With Sampling and Approximate Pre-Aggregation. In *Proceedings .of the International Conference on Very Large Databases*, 886-897.

JERMAINE C., POL A. AND ARUMUGAM S. 2004. Online Maintenance of Very Large Random Samples. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004.

KEMPE D., DOBRA A. AND GEHRKE J. 2003. Gossip-Based Computation of Aggregate Information. *In the Proceedings of IEEE Foundations of Computer Science,* 2003, 482-491.

KOUDAS N. AND SRIVASTAVA D. 2003. Data Stream Query Processing: A Tutorial. *Tutorial at the International Conference on Very Large Databases*, 2003.

LOHR, S. 1999. *Sampling: Design and Analysis*. Duxbury Press.

MILLER R. G., JR. 1981. *Simultaneous Statistical Inference*. Springer Series in Statistics.

MITCHELL T. 1997. *Machine Learning*. McGraw Hill.

MOTWANI R. AND RAGHAVAN P. 1995. *Randomized Algorithms*. Cambridge University Press.

OLKEN F. 1993. Random Sampling from Databases. *Ph.D. Dissertation*, Computer Science, UC Berkeley.

PAVLOV D., MANNILA H, AND SMYTH P. 2003. Beyond Independence: Probabilistic Methods for Query Approximation on Binary Transaction Data. *IEEE Transactions on Knowledge and Data Engineering*, 15(6) 1409-1421.

POLYZOTIS N. GAROFALAKIS M. N. AND IOANNIDIS Y. 2004. Approximate XML Query Answers. In *Proceedings of the SIGMOD International Conference on Management of Data*, 2004.

POOSALA V. AND GANTI V. 1999. Fast Approximate Answers to Aggregate Queries on a Data Cube. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, 1999.

THISTED R. A. 1988. *Elements of Statistical Computing*. Chapman and Hall.

TPC Benchmark R. Decision Support. Revision 1.1.0. http://www.tpc.org.

VALLIANT R. AND J. GENTLE. 1997. An Application of Mathematical Programming to a Sample Allocation Problem. *Computational Statistics and Data Analysis*, 25, 337-360.

VITTER J. 1985. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1).

VITTER J. AND WANG M. 1999. Approximate Computation of Multidimensional Aggregates of Sparse Data using Wavelet. In *Proceedings of the ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*, 1999.

VITTER J., WANG M. AND IYER B. 1998. *Data Cube Approximation and Histogram via Wavelets*. In *Proceedings of the International Conference on Information and Knowledge Management*, 1998.

ZIPF G.E. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press Inc.

APPENDIX: PROOFS OF MATHEMATICAL RESULTS

In this appendix, we provide the missing proofs of some of the mathematical results stated earlier. We first review the well-known concept of *Chernoff bounds* from probability theory. We then prove Lemma 6, and followed by Lemmas 4, 7 and 8 in that order.

Binomial Distributions and Chernoff Bounds

Consider a coin with bias β (i.e., when tossed the probability of head is β). Let *m* be the number of heads that occur when the coin is tossed independently *n* times. If we view *m* as a random variable, its probability distribution is the well-known (e.g., see Motwani and Raghavan [1995]) *binomial distribution* $p(m) = \binom{n}{m} \beta^m (1-\beta)^{n-m}$. The expected value of *m* is $E[m] = \beta n$. The variance of *m* is $E\left[(m-\beta n)^2\right] = \beta(1-\beta)n$. As *n* gets large, it is known that the probability distribution gets tightly concentrated around its mean. This is quantified by the following *Chernoff bound* (where *c* is a constant):

$$\forall \varepsilon \in [0,1], p((1-\varepsilon)\beta n < m < (1+\varepsilon)\beta n) \geq 1 - 2e^{-c\varepsilon^2 \beta n}$$

See Motwani and Raghavan [1995] for discussion of Chernoff bounds.

Proof of Lemma 6

We assume that δ, γ, r, k_1, k_2, …, k_r are all constants, while n, n_1, n_2, …, n_r may vary. Q will always represent a fixed COUNT query from the workload, whereas Q' will always represent an incoming query drawn randomly from the distribution $p_{\{Q\}}$. The sets $R_Q \subseteq R$ and R' $\subseteq$ R will represent the sets of records selected by Q and Q' respectively. Recall from Section 3 that the distribution $p_{\{Q\}}$ actually maps subsets of R to probabilities. Thus, in the rest of the proof we will sometimes view a query simply as the subsets of records it selects (e.g. Q' and R' may be used interchangeably).

Since we view queries as subsets of records, we see that *MSE*($p_{\{Q\}}$)is the expected value of *SE*(R') for a subset R' randomly drawn from the distribution $p_{\{Q\}}$ (recall the exact definitions of squared error and mean squared error from Section 2). That is,

$$MSE(p_{\{Q\}}) = \sum_{R' \subseteq R} p_{\{Q\}}(R') \cdot SE(R') \tag{3}$$

Our task is to expand and simplify the RHS of Equation (3) and show that it is approximately equal to *ApproxMSE*($p_{\{Q\}}$).

We first partition the set of all 2^n subsets of R into a (large) number of groups as follows. Consider *r* integers $m_1, m_2, \ldots, m_r$, such that $0 \le m_j \le n_j$. These integers define a group

$$G(m_1, m_2, \ldots, m_r) = \{R' \mid m_j = |R' \cap R_j|, 1 \le j \le r\}$$

That is, G($m_1, m_2, \ldots, m_r$) contains all subsets of R that select exactly m_1 records from R_1, m_2 records from R_2, and so on. Clearly, the number of groups is $\prod_{1 \le j \le r} (n_j + 1)$

We can also derive the size of each group (i.e., the number of subsets in each group):

$$|G(m_1, m_2, \ldots, m_r)| = \prod_{1 \le j \le r} \binom{n_j}{m_j} \qquad (4)$$

Consider a subset R' that belong to any given group, say G($m_1, m_2, \ldots, m_r$). We show that the squared error *SE*(R') may be derived as follows.

$$SE(R') = \sum_{1 \le j \le r} \frac{n_j^2}{k_j} \frac{m_j}{n_j} \left(1 - \frac{m_j}{n_j}\right) \Bigg/ \left(\sum_{1 \le j \le r} m_j\right)^2$$

Consider the *j*th term in the numerator. It represents the expected squared error in estimating the count of (R'∩Rj), i.e., in estimating the sum of the portion of $POP_{Q'}$ that corresponds to R_j (see Section 4.1 for a definition of $POP_{Q'}$). This portion of $POP_{Q'}$ may be viewed as a population of size n_j, with m_j 1's and $n_j - m_j$ 0's. It is easy to derive the variance of such a population to be $\frac{m_j}{n_j}\left(1 - \frac{m_j}{n_j}\right)$. Since each region has k_j samples allocated to it, we can use Lemma 2 from Section 2 to show that the entire numerator represents the squared error in estimating the count of R'. The denominator represents the expected count of R'. Thus the ratio represents the relative squared error in estimating the count of R'.

Our task is to show that the expected value of *SE*(R') approaches *ApproxMSE*($p_{\{Q\}}$) in the limit when *n* tends to infinity. Note that *SE*(R') equals *ApproxMSE*($p_{\{Q\}}$) if we replace each m_j by $\delta * n_j$ or $\gamma * n_j$, depending on whether Rj is inside or outside R_Q. It is easy to see that *SE*(R') is the same for each subset R' in the group G($m_1, m_2, \ldots, m_r$). Let us denote this as *SE*($m_1, m_2, \ldots, m_r$).

$$SE(m_1, m_2, \ldots, m_r) = \sum_{1 \le j \le r} \frac{n_j^2}{k_j} \frac{m_j}{n_j} \left(1 - \frac{m_j}{n_j}\right) \Bigg/ \left(\sum_{1 \le j \le r} m_j\right)^2 \qquad (5)$$

Our next task is to expand $p_{\{Q\}}$(R') for any given subset R' of group G(m_1, m_2, …, m_r). Using Equation 1 in Section 3, we can derive $p_{\{Q\}}$(R') as follows.

$$p_{\{Q\}}(R') = \delta^{\left(\sum_{R_j \subseteq R_Q} m_j\right)} (1-\delta)^{\left(\sum_{R_j \subseteq R_Q} n_j - m_j\right)} \gamma^{\left(\sum_{R_j \subseteq R \setminus R_Q} m_j\right)} (1-\gamma)^{\left(\sum_{R_j \subseteq R \setminus R_Q} n_j - m_j\right)} \tag{6}$$

Note that $p_{\{Q\}}$(R') is the same for each subset R' in the group G(m_1, m_2, …, m_r). Let p(m_1, m_2, …, m_r) denote the probability that a randomly drawn subset R' belongs to G(m_1, m_2, …, m_r). Using Equations (4) and (6), it is easy to see that $p(m_1, m_2, ..., m_r) =$

$$\left[\prod_{1 \le j \le r} \binom{n_j}{m_j}\right] \left[\delta^{\left(\sum_{R_j \subseteq R_Q} m_j\right)} (1-\delta)^{\left(\sum_{R_j \subseteq R_Q} n_j - m_j\right)} \gamma^{\left(\sum_{R_j \subseteq R \setminus R_Q} m_j\right)} (1-\gamma)^{\left(\sum_{R_j \subseteq R \setminus R_Q} n_j - m_j\right)} \right]$$

By rearranging factors we get $p(m_1, m_2, ..., m_r) =$

$$\left[\prod_{R_j \subseteq R_Q} \binom{n_j}{m_j} \delta^{m_j} (1-\delta)^{n_j - m_j}\right] \left[\prod_{R_j \subseteq R \setminus R_Q} \binom{n_j}{m_j} \gamma^{m_j} (1-\gamma)^{n_j - m_j}\right] \tag{7}$$

Observe that in the above equation the fundamental regions inside R_Q are treated differently from the fundamental regions that are outside R_Q. In the interest of uniformity, we adopt the following notation. Let each fundamental region R_j be associated with a parameter β_j, such that if R_j is inside R_Q then $\beta_j = \delta$ and if R_j is outside R_Q then $\beta_j = \gamma$. Equation (7) may then be rewritten as

$$p(m_1, m_2, ..., m_r) = \prod_{1 \le j \le r} \binom{n_j}{m_j} \beta_j^{m_j} (1-\beta_j)^{n_j - m_j} \tag{8}$$

The RHS is the probability that a random subset R' will select exactly m_j records from R_j. Let us denote this as $p(m_j)$. If we view m_j as a random variable, $p(m_j)$ is a binomial distribution with a mean of $\beta * n_j$. Thus p(m_1, m_2, …, m_r) is simply a product of different binomial distributions:

$$p(m_1, m_2, ..., m_r) = \prod_{1 \le j \le r} p(m_j)$$

Let C be the Cartesian product $\{0..n_1\} \times \{0..n_2\} \times ... \times \{0..n_r\}$. Equation (3) may be rewritten as

$$MSE(p_{\{Q\}}) = \sum_{[m_1,m_2,...,m_r]\in C} p(m_1, m_2,..., m_r) * SE(m_1, m_2,..., m_r)$$

Using Equations (5) and (8), we get $MSE(p_{\{Q\}}) =$

$$\sum_{[m_1,m_2,...,m_r]\in C} \left[\prod_{1\le j\le r} \binom{n_j}{m_j} \beta_j^{m_j} (1-\beta_j)^{n_j-m_j} \right] \left[\frac{\sum_{1\le j\le r} \frac{n_j^2}{k_j} \frac{m_j}{n_j} \left(1 - \frac{m_j}{n_j}\right)}{\left(\sum_{1\le j\le r} m_j\right)^2} \right]$$

Let us label a fundamental region R$_j$ as *small* if $0 < n_j < \sqrt{n}$, and *large* otherwise (i.e., if $\sqrt{n} \le n_j \le n$). For each large fundamental region R$_j$, define quantity $\varepsilon_j = \ln n_j \big/ \sqrt{n_j}$.

Let C$_1$ ⊆ C be the cartesian product defined as $\{l_1..u_1\} \times \{l_2..u_2\} \times ... \times \{l_r..u_r\}$, where if R$_j$ is small then $l_j = 0$ and $u_j = n_j$, and if R_j is large then $l_j = (1 - \varepsilon_j)\beta_j n_j$ and $u_j = (1+ \varepsilon_j)\beta_j n_j$. Let C$_2$ be the set of vectors defined as C – C$_1$.

Define MSE_1 and MSE_2 as follows:

$$MSE_1 = \sum_{[m_1,m_2,...,m_r]\in C_1} p(m_1, m_2,..., m_r) * SE(m_1, m_2,..., m_r)$$
$$MSE_2 = \sum_{[m_1,m_2,...,m_r]\in C_2} p(m_1, m_2,..., m_r) * SE(m_1, m_2,..., m_r) \qquad (9)$$

Clearly we have $MSE(p_{\{Q\}}) = MSE_1 + MSE_2$. We shall first show that in the limit when n tends to infinity, MSE_2 goes to 0. If we examine Equation (5), we can derive a crude (but simple) upper bound for $SE(m_1, m_2, …, m_r)$ as:

$$\frac{\sum_{1\le j\le r} \frac{n_j^2}{k_j} \frac{m_j}{n_j} \left(1 - \frac{m_j}{n_j}\right)}{\left(\sum_{1\le j\le r} m_j\right)^2} \le \frac{\sum_{1\le j\le r} \frac{n_j^2}{k_j} \frac{m_j}{n_j} \left(1 - \frac{m_j}{n_j}\right)}{\sum_{1\le j\le r} m_j^2}$$

$$\le \frac{\sum_{1\le j\le r} n_j m_j}{\sum_{1\le j\le r} m_j^2} \le \frac{\sum_{1\le j\le r} n m_j}{\sum_{1\le j\le r} m_j^2} \le \frac{\sum_{1\le j\le r} n m_j}{\sum_{1\le j\le r} m_j} \le n$$

Thus we get $MSE_2 \leq \sum_{[m_1,m_2,\ldots,m_r]\in C_2} p(m_1,m_2,\ldots,m_r) * n =$

$$= \sum_{[m_1,m_2,\ldots,m_r]\in C_2} \left[\prod_{1\leq j\leq r} \binom{n_j}{m_j} \beta_j^{\,m_j} (1-\beta_j)^{n_j-m_j} \right] * n$$

$$= \left[1 - \sum_{[m_1,m_2,\ldots,m_r]\in C_1} \left[\prod_{1\leq j\leq r} \binom{n_j}{m_j} \beta_j^{\,m_j} (1-\beta_j)^{n_j-m_j} \right] \right] * n$$

$$= \left[1 - \prod_{1\leq j\leq r} \sum_{l_j<m_j<u_j} \binom{n_j}{m_j} \beta_j^{\,m_j} (1-\beta_j)^{n_j-m_j} \right] * n$$

Using Chernoff bounds, the above is

$$\leq \left[1 - \prod_{l\arg eR_j} \left(1-2e^{-c\varepsilon_j{}^2\beta_j n_j}\right) \right] * n = \left[1 - \prod_{l\arg eR_j} \left(1-2e^{-c\beta_j \ln^2 n_j}\right) \right] * n$$

$$\leq \left[1 - \left(1-2e^{-c\min\{\delta,\gamma\}\ln^2\sqrt{n}}\right)^r \right] * n = \left[1 - \left(1-2e^{-0.25c\min\{\delta,\gamma\}\ln^2 n}\right)^r \right] * n$$

$$= \frac{2\binom{r}{1}n}{e^{0.25c\min\{\delta,\gamma\}\ln^2 n}} - \frac{2^2\binom{r}{2}n}{e^{2(0.25c)\min\{\delta,\gamma\}\ln^2 n}} + \ldots$$

$$+ (-1)^{j+1} \frac{2^j\binom{r}{j}n}{e^{j(0.25c)\min\{\delta,\gamma\}\ln^2 n}} + \ldots + (-1)^{r+1} \frac{2^r\binom{r}{r}n}{e^{r(0.25c)\min\{\delta,\gamma\}\ln^2 n}}$$

It is quite easy to show that as *n* tends to infinity, each of the above terms goes to zero. To see this, consider the *j*th term. If we take the natural log of the numerator, we get an expression that is linear in ln(*n*), whereas if we take the natural log of the denominator, we get an expression that is quadratic in ln(*n*). Thus we conclude that

$$\lim_{n\to\infty} MSE_2 = 0 \qquad (10)$$

Next, we turn our attention to MSE_1. We shall show that when *n* tends to infinity, the ratio $\dfrac{MSE_1}{ApproxMSE\left(p_{\{Q\}}\right)} \to 1$

Recall the definition of *ApproxMSE*($p_{\{Q\}}$) from the statement of the lemma. Using β_j instead of δ and γ, we can rewrite it as follows.

$$ApproxMSE\,(p_{\{Q\}}) = \sum_{1\le j\le r} \frac{n_j^{\;2}}{k_j}\,\beta_j\left(1-\beta_j\right) \Bigg/ \left(\sum_{1\le j\le r}\beta_j n_j\right)^2 \tag{11}$$

Using Equations (5), (9) and (11), we get $\lim\limits_{n\to\infty} \dfrac{MSE_1}{ApproxMSE(p_{\{Q\}})}$

$$= \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p\left(m_1, m_2,\ldots, m_r\right) \left[\frac{\displaystyle\sum_{1\le j\le r}\frac{n_j^{\;2}}{k_j}\frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\displaystyle\sum_{1\le j\le r}\frac{n_j^{\;2}}{k_j}\beta_j\left(1-\beta_j\right)}\right]\left[\frac{\displaystyle\sum_{1\le j\le r}\beta_j n_j}{\displaystyle\sum_{1\le j\le r} m_j}\right]^2$$

We first show that we only need be concerned with large fundamental regions. Dividing each factor in the numerator and denominator by n^2, we get

$$= \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p\left(m_1, m_2,\ldots, m_r\right) \left[\frac{\displaystyle\sum_{1\le j\le r}\left(\frac{n_j}{n}\right)^2\frac{1}{k_j}\frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\displaystyle\sum_{1\le j\le r}\left(\frac{n_j}{n}\right)^2\frac{1}{k_j}\beta_j\left(1-\beta_j\right)}\right]\left[\frac{\displaystyle\sum_{1\le j\le r}\beta_j\left(\frac{n_j}{n}\right)}{\displaystyle\sum_{1\le j\le r}\left(\frac{m_j}{n}\right)}\right]^2$$

For small fundamental regions R$_j$, we know that in the limit when n tends to infinity, both n_j/n and m_j/n tend to zero. Thus in the limit the above expression reduces to

$$= \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p\left(m_1, m_2,\ldots, m_r\right) \left[\frac{\displaystyle\sum_{l\arg eRj}\left(\frac{n_j}{n}\right)^2\frac{1}{k_j}\frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\displaystyle\sum_{l\arg eRj}\left(\frac{n_j}{n}\right)^2\frac{1}{k_j}\beta_j\left(1-\beta_j\right)}\right]\left[\frac{\displaystyle\sum_{l\arg eRj}\beta_j\left(\frac{n_j}{n}\right)}{\displaystyle\sum_{l\arg eRj}\left(\frac{m_j}{n}\right)}\right]^2$$

Multiplying each factor in the numerator and denominator by n^2, we get back

$$= \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p\left(m_1, m_2,\ldots, m_r\right) \left[\frac{\displaystyle\sum_{l\arg eRj}\frac{n_j^{\;2}}{k_j}\frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\displaystyle\sum_{l\arg eRj}\frac{n_j^{\;2}}{k_j}\beta_j\left(1-\beta_j\right)}\right]\left[\frac{\displaystyle\sum_{l\arg eRj}\beta_j n_j}{\displaystyle\sum_{l\arg eRj} m_j}\right]^2 \tag{12}$$

We know that for each large fundamental region, (1− ε_j)$\beta_j n_j \le m_j \le$ (1+ ε_j)$\beta_j n_j$. We first calculate an upper bound for the limit. The RHS of Eqn. (12) can be upper-bounded as

$$\le \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p(m_1,m_2,\ldots,m_r)\left[\frac{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\frac{(1+\varepsilon_j)\beta_j n_j}{n_j}\left(1-\frac{(1-\varepsilon_j)\beta_j n_j}{n_j}\right)}{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\beta_j(1-\beta_j)}\right]\left[\frac{\sum\limits_{l\arg eRj}\beta_j n_j}{\sum\limits_{l\arg eRj}(1-\varepsilon_j)\beta_j n_j}\right]^2$$

$$\le\left[\frac{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}(1+\varepsilon_j)\beta_j\left(1-(1-\varepsilon_j)\beta_j\right)}{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\beta_j(1-\beta_j)}\right]\left[\frac{\sum\limits_{l\arg eRj}\beta_j n_j}{\sum\limits_{l\arg eRj}(1-\varepsilon_j)\beta_j n_j}\right]^2 \tag{13}$$

For each large R$_j$, it is easy to see that $\lim\limits_{n\to\infty}\varepsilon_j = \lim\limits_{n\to\infty}\frac{\ln n_j}{\sqrt{n_j}} = \lim\limits_{n_j\to\infty}\frac{\ln n_j}{\sqrt{n_j}} = 0$

Thus, in the limit the *j*th terms in the numerator will approach the corresponding *j*th terms in the denominator of Equation (13). Thus we have

$$\lim_{n\to\infty}\frac{MSE_1}{ApproxMSE\,(p_{\{Q\}})}\le 1 \tag{14}$$

We next calculate a lower bound for RHS of Equation (14). The RHS can be lower-bounded as

$$\ge \sum_{[m_1,m_2,\ldots,m_r]\in C_1} p(m_1,m_2,\ldots,m_r)\left[\frac{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\frac{(1-\varepsilon_j)\beta_j n_j}{n_j}\left(1-\frac{(1+\varepsilon_j)\beta_j n_j}{n_j}\right)}{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\beta_j(1-\beta_j)}\right]\left[\frac{\sum\limits_{l\arg eRj}\beta_j n_j}{\sum\limits_{l\arg eRj}(1+\varepsilon_j)\beta_j n_j}\right]^2$$

$$\ge \prod_{l\arg eR_j}\left(1-2e^{-c\beta_j\ln^2 n_j}\right)\left[\frac{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}(1-\varepsilon_j)\beta_j\left(1-(1+\varepsilon_j)\beta_j\right)}{\sum\limits_{l\arg eRj}\frac{n_j^{\,2}}{k_j}\beta_j(1-\beta_j)}\right]\left[\frac{\sum\limits_{l\arg eRj}\beta_j n_j}{\sum\limits_{l\arg eRj}(1+\varepsilon_j)\beta_j n_j}\right]^2 \tag{15}$$

For each large fundamental region, we note that

$$\lim_{n\to\infty}\left(1-2e^{-c\beta_j \ln^2 n_j}\right)=1-\lim_{n_j\to\infty}2e^{-c\beta_j \ln^2 n_j}=1-0=1$$

As for the remaining portion of Equation (15), in the limit the *j*th terms in the numerator will approach the corresponding *j*th terms in the denominator of Equation (15). Equation (15) thus reduces to

$$\lim_{n\to\infty}\frac{MSE_2}{ApproxMSE(p_{\{Q\}})}\geq 1 \tag{16}$$

Combining Equations (10), (14) and (16), we have

$$\lim_{n\to\infty}\frac{MSE(p_{\{Q\}})}{ApproxMSE(p_{\{Q\}})}=1$$

This concludes the proof of Lemma 6. ■

Proof of Lemma 4

Recall that the stratification F = {R$_1$, R$_2$, …, R$_r$} is defined as the set of fundamental regions. Suppose the optimal stratification was actually a different stratification, G = {G$_1$, G$_2$... G$_s$}. Consider the stratification H = {H$_1$, H$_2$, ….H$_u$} = {R$_j$∩G$_i$ | 1≤ *j* ≤ *r*, 1 ≤ *i* ≤ *s*}. Since stratifying an optimal stratification any further does not reduce the *MSE*, H also represents an optimal stratification.

We first describe the simple case where the workload consists of a single query Q. Let us use the notation $MSE_E(p_{\{Q\}})$ to denote $MSE(p_{\{Q\}})$ for a stratification E. We shall show that the minimum value of $MSE_H(p_{\{Q\}})$ is asymptotically the same as the minimum value of $MSE_F(p_{\{Q\}})$. $MSE_H(p_{\{Q\}})$ can be asymptotically calculated according to Lemma 6 (even though Lemma 6 was proven for the stratification F, it can be extended for any stratification that represents a further stratification of F, such as H). Let *h*$_1$, *h*$_2$, …, *h*$_u$ be the optimal allocation of the *k* samples in the strata H$_1$, H$_2$, …., H$_u$ respectively. By Lemma 6, we get

$$\lim_{n\to\infty}MSE_H(p_{\{Q\}})=\frac{\sum_{H_j\in H}\frac{|H_j|^2}{h_j}\beta_j(1-\beta_j)}{\left(\sum_{H_j\in H}\beta_j|H_j|\right)^2}$$

In minimizing $MSE_{\mathrm{H}}(p_{\{\mathrm{Q}\}})$ we can ignore the denominator. Thus is equivalent to minimizing the expression $\sum_{H_j \in H} \frac{\left|H_j\right|^2 \beta_j \left(1-\beta_j\right)}{h_j}$

Using techniques similar to those in Lemma 7 (to be proved next), we see that this gets minimized when each $h_j = k\left(\frac{\left|H_j\right|\sqrt{\beta_j\left(1-\beta_j\right)}}{\sum_{H_i \in H}\left|H_i\right|\sqrt{\beta_i\left(1-\beta_i\right)}}\right)$.Plugging these values back in $MSE_{\mathrm{H}}(p_{\{\mathrm{Q}\}})$, we see that the minimum value of $MSE_{\mathrm{H}}(p_{\{\mathrm{Q}\}})$ is asymptotically equal to

$$\min MSE_H(p_Q) = \frac{\left(\sum_{H_j \in H}\left|H_j\right|\sqrt{\beta_j\left(1-\beta_j\right)}\right)^2}{k\left(\sum_{H_j \in H}\beta_j\left|H_j\right|\right)^2}$$

Now let us consider the stratification F. Let k_1, k_2, …, k_{r} be the optimal allocation of the k samples in the fundamental regions R$_1$, R$_2$, …, R$_{\mathrm{r}}$ respectively. Similar to above, we see that $MSE_{\mathrm{F}}(p_{\{\mathrm{Q}\}})$ gets minimized when each $k_j = k\left(\frac{n_j\sqrt{\beta_j\left(1-\beta_j\right)}}{\sum_{R_i \in F} n_j\sqrt{\beta_i\left(1-\beta_i\right)}}\right)$.Plugging these values back, we can similarly derive the minimum value of $MSE_{\mathrm{F}}(p_{\{\mathrm{Q}\}})$ to be asymptotically equal to $\frac{\left(\sum_{R_j \in F} n_j\sqrt{\beta_j\left(1-\beta_j\right)}\right)^2}{k\left(\sum_{R_j \in F}\beta_j n_j\right)^2}$

We observe the following points. Since H represents a further stratification of F, each stratum of F wholly contains a set of strata of H. Furthermore, the βj associated with any stratum Rj of F is the same for each stratum of H that is contained within Rj. We thus get

$$\min MSE_F(p_Q) = \frac{\left(\sum_{R_j \in F} n_j \sqrt{\beta_j(1-\beta_j)}\right)^2}{k\left(\sum_{R_j \in F} \beta_j n_j\right)^2} = \frac{\left(\sum_{R_j \in F}\left(\sum_{H_i \in R_j} |H_i|\right)\sqrt{\beta_j(1-\beta_j)}\right)^2}{k\left(\sum_{R_j \in F}\left(\sum_{H_i \in R_j} |H_i|\right)\beta_j\right)^2}$$

$$= \frac{\left(\sum_{H_j \in H} |H_j| \sqrt{\beta_j(1-\beta_j)}\right)^2}{k\left(\sum_{H_j \in H} \beta_j |H_j|\right)^2} = \min MSE_H(p_Q)$$

The above arguments can be extended to the case when the workload contains more than one query. This concludes the proof of Lemma 4.■

Proof of Lemma 7

We first eliminate one of the variables, say k_r, by replacing it with $k - (k_1 + \ldots + k_{r-1})$. If we partially differentiate $\sum_{1 \le j \le r} \frac{\alpha_j}{k_j}$ by $k_1, \ldots, k_{r-1}$ respectively and set each derivative to zero, this results in r – 1 equations, where the jth equation is $\frac{\alpha_j}{k_j^2} = \frac{\alpha_r}{\left(k - \sum_{1 \le j \le r-1} k_j\right)^2}$. Taking square roots and simplifying further, we see that the optimal value of k_j is proportional to $\sqrt{\alpha_j}$. The lemma then follows. We note that the proof is along similar lines to other well-known methods for minimizing functions of the form $\sum_{1 \le j \le r} \frac{\alpha_j}{k_j}$ that arise in different contexts (e.g. Acharya et al [1999]; Cochran [1977]). ■

Proof of Lemma 8

As in the earlier proof of Lemma 6, we assume that δ, γ, r, k_1, k_2, …, k_r are all constants, while n, n_1, n_2, …, n_r may vary. Q will always represent a fixed query from the workload, whereas Q' will always represent an incoming query drawn randomly from the distribution $p_{\{Q\}}$. Also recall that we view R as being partitioned into $h*r$ strata, where each stratum R$_j$ has n_j records, each with the same aggregate column value y_j (which is positive). Since the proof is very similar to that of Lemma 6, we only highlight the important differences. Our notation will be similar to that used in Lemma 7. The equation corresponding to Equation (5) may be easily derived as

$$SE(m_1, m_2, ..., m_{h*r}) = \frac{\sum_{1\le j\le h*r} \frac{n_j^2 y_j^2}{k_j} \frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\left(\sum_{1\le j\le h*r} m_j y_j\right)^2} \tag{17}$$

As before, let us define MSE_1 and MSE_2 such that $MSE(p_{\{Q\}}) = MSE_1 + MSE_2$
We shall first show that in the limit when n tends to infinity, MSE_2 goes to 0. If we examine Equation (17), we can derive a crude (but simple) upper bound for $SE(m_1, m_2, \ldots, m_{h*r})$ as follows:

$$SE(m_1, m_2, ..., m_{h*r}) = \frac{\sum_{1\le j\le h*r} \frac{n_j^2 y_j^2}{k_j} \frac{m_j}{n_j}\left(1-\frac{m_j}{n_j}\right)}{\left(\sum_{1\le j\le h*r} m_j y_j\right)^2} \le \frac{\sum_{1\le j\le h*r} n_j m_j y_j^2}{\sum_{1\le j\le h*r} m_j^2 y_j^2}$$

$$\le \frac{\sum_{1\le j\le h*r} n m_j y_j^2}{\sum_{1\le j\le h*r} m_j^2 y_j^2} \le \frac{\sum_{1\le j\le h*r} n m_j y_j^2}{\sum_{1\le j\le h*r} m_j y_j^2} \le n$$

Important: Note that in the above derivation (especially in the denominator of the first simplification), it is critical that all the y_j's are of the same sign (i.e., either all positive or all negative). If this were not the case, it may be possible that the positive and negative y_j's may cancel each other, leading to very small denominators, and consequently very large squared errors.

The rest of the arguments are very similar to Lemma 6. Intuitively, as in Lemma 6, the samples allocated from each stratum are in charge of estimating a count of the rows selected from that stratum – the only difference that the counted is weighted by a factor of y_{j}. We omit the straightforward details, but mention that it can be shown that when n tends to infinity, both $MSE_2 \to 0$ and $\frac{MSE_1}{ApproxMSE\left(p_{\{Q\}}\right)} \to 1$. Thus we have

$$\lim_{n\to\infty} \frac{MSE(p_{\{Q\}})}{ApproxMSE(p_{\{Q\}})} = 1$$

This concludes the proof of Lemma 8. ■