A Handbook for Building an Approximate Query Engine

Barzan Mozafari Ning Niu
University of Michigan, Ann Arbor
{mozafari,nniu}@umich.edu

Abstract

There has been much research on various aspects of Approximate Query Processing (AQP), such as different sampling strategies, error estimation mechanisms, and various types of data synopses. However, many subtle challenges arise when building an actual AQP engine that can be deployed and used by real world applications. These subtleties are often ignored (or at least not elaborated) by the theoretical literature and academic prototypes alike. For the first time to the best of our knowledge, in this chapter, we focus on these subtle challenges that one must address when designing an AQP system. Our intention for this chapter is to serve as a handbook listing critical design choices that database practitioners must be aware of when building or using an AQP system, not to prescribe a specific solution to each challenge.

1 Introduction

We start this chapter with a discussion of when and why AQP is a viable solution in Section 2. After a brief overview of the typical anatomy of AQP systems in Section 3, we discuss the common sampling techniques used in practice in Section 4. Often, database developers are not aware of the subtle differences between different sampling strategies and their implications on the error estimation procedures. While confidence intervals are commonly used for quantifying the approximation error, in Section 5 we discuss other important types of error (i.e., subset and superset errors) that are not expressible as confidence intervals, but are quite common in practice. We then discuss another source of error, *bias*, in Section 6. Surprisingly, while well-studied and understood by statisticians, bias seems to be often overlooked in the database literature on AQP systems. The complexity and variety of different types of errors can easily render an AQP system incomprehensible and unusable for an average user of database systems. Intuitive but truthful communication of error with the end user is discussed in Section 7. We conclude this chapter in Section 8.

2 Why use Approximation in the First Place?

Since its inception 25 years ago [63], Approximate Query Processing (AQP) has always been an active area of research in academia [7, 11, 20, 25, 32, 53, 58]. However, AQP has recently gained substantial attention in the commercial world as well. Soon after the commercialization of BlinkDB by Databricks [1], other Big Data query engines also started to add support for approximation features in their query engines (e.g., SnappyData [5]

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

or Facebook's Presto [4]). In the rest of this section, we try to explain the reason, rationale, and justification behind this commercial interest.

2.1 The Losing Battle for Interactive Response Times

Online services, wireless devices, and scientific simulations are all creating massive volumes of data at unprecedented rates. This abundance of rich datasets has made *data-driven decisions* the predominant approach across businesses, science, and even governments. Ironically, existing data processing tools have now become the bottleneck in data-driven activities. When faced with sufficiently large datasets (e.g., a few terabytes), even the fastest database systems can take hours or days to answer the simplest queries [11]. This response time is simply unacceptable to many users and applications. The exploratory data analytics is often an *interactive* and *iterative* process: a data scientist forms an initial hypothesis (e.g., by visualizing the data), consults the data, adjusts his/her hypothesis accordingly, and repeats this process until a satisfactory answer is discovered. Thus, slow and costly interactions with data can severely inhibit a data scientist's productivity, engagement, and even creativity.[1]

Driven by the growing market for interactive analytics, commercial and open source data warehouses (e.g., Teradata, HP Vertica, Hive, Impala, Presto, Spark SQL, Redshift) have entered an arms race towards providing interactive response times through implementing various optimization techniques. These optimization range from exploiting parallelism, to indexing, materialization, better query plans, data compression, columnar formats, and in-memory and in-situ processing. In essence, these optimizations are no different than the mainstream database research on query processing over the past four decades, where the goal has been simply to:

1. efficiently access *all* relevant tuples to the query while avoiding the irrelevant tuples (e.g., via indexing, materialization, compression, caching); and

2. choose a query plan with the least cost among a set of alternative plans that are all *equivalent* (i.e., all plans are correct).

As long as we pursue these utopian goals,[2] all our query optimization techniques are eventually destined to fail. This is because the growth rate of data has already surpassed Moore's law [3]. It is true that the set of tuples queried by the user, called the *working set*, is often significantly smaller than the entire data, e.g., queries might focus on the last week worth of data. However, even the data generated and collected each week is growing at the same speed. This means that even if we know which exact tuples are accessed by a query in advance, at some point, the number of those tuples will be large enough that they will no longer fit in memory (even when compressed). In other words, since memory and CPU improvements are lagging *exponentially* behind the rate of data growth, our ability to provide interactive response times will only decrease. Eventually, when the datasets become large enough, no optimization technique will be able to provide interactive response times unless we relax our goal of processing *all* relevant tuples.

Therefore, AQP—whereby only a small fraction of the relevant tuples are processed—presents itself as the only viable approach to ensure interactive response times in the near future.

2.2 Perfect Decisions are Possible even with Imperfect Answers

In many applications, query results are only useful inasmuch as they enable best decisions, as exemplified by the following use cases.

[1] For instance, studies have shown that to maintain a human's interactive engagement with a computer, the response time must not exceed X–Y seconds.

[2] Our discussion here is focused on relaxing the first goal. For a discussion on relaxing the second goal refer to [46].

A/B testing —. The goal in A/B testing is to compare two quantities $f(A)$ and $f(B)$ for some assessment function f, and thereby choose the better explanation. However, as long as our estimates of these quantities reliably allow for such comparisons, there is no need for computing their exact values. For instance, as long as our approximations guarantee with high confidence (e.g., 99%) that $f(A) \in [0.1, 0.2]$ while $f(B) \in [0.5, 0.8]$, there is no need for knowing the exact values of those two quantities.

Hypothesis testing — A major goal of data scientists is to evaluate and test a set of competing hypotheses and choose the *best* one. Choosing the best hypothesis often involves comparing the effectiveness scores of different hypotheses. The effectiveness score of a hypothesis can be measured by computing some function of the hypothesis, e.g., probability values (p-values), the generalization-error, confidence intervals, occurrence ratios, or reduction of entropy. These scores are initially only useful insofar as they allow for ruling out the unlikely hypotheses in light of the more likely ones. In other words, scientists do not need the precise value of these scores. As long as we can guarantee that a hypothesis yields a lower score than others, the researchers can proceed to choose the better hypothesis.

Root cause analysis — Scientists and data analysts frequently seek explanations for or identify the root causes of various observations, symptoms, or faults. Similar to previous cases, root cause analysis can be accomplished without having precise values.

Exploratory analytics — Data scientists often slice and dice their dataset in search of interesting trends, correlations or outliers. For these exploration purposes, receiving approximate answers at interactive speeds is preferred over exact and slow results. In fact, research on human-computer interaction has shown that, to keep users engaged, the response time of an interactive system must be below 10 seconds. In particular, during creative efforts, the idle time beyond a couple of seconds, while the user waits to see the consequence of his/her query, is inhibitive and intolerable [44].

Feature selection — An important step in training high-quality models in machine learning consists of searching the space of possible subsets of features. The exponential number of possibilities has made feature selection one of the most expensive (yet crucial) steps in a machine learning workflow. Similar to the previous examples, as long as a set of features can be assessed as inferior to another set, the former can be safely ruled out without having to compute the exact prediction error of the model under these subsets of features (e.g., see [38]).

Big data visualization —Interactive visualization is a key means through which analysts discover interesting trends in scientific or business-related data. Modern visualization tools, such as Tableau [6] and GoodData [2], use external database engines to manage large datasets. When the user requests a plot, the visualization tool submits a query to the database engine to fetch the data needed for the visualization. Unfortunately, for large datasets it may take the system several minutes or even hours to obtain the tuples necessary for generating a plot. Increasingly, the explicit goal of visualization systems is not just to generate a single precise image, but to enable a fluid back-and-forth engagement between the user and the dataset [41]. When the database system fails to respond in a short amount of time, this fluid interactivity is lost. However, loading (and plotting) more data does not always improve the quality of the visualization. This is due to three reasons [55]: (i) the finite number of pixels on the screen, (ii) the cognitive limitations of human vision in noticing small details, and (iii) the redundancy in most real-world datasets. Thus, it is often possible for the database system to generate a degraded query answer using a small sample of the large dataset but with minimal impact on the visualization's quality [27, 37, 55].

2.3 The Data Itself is Often Noisy

Most of the data collected in the real world is inherently noisy, due to human error, missing values, white noise, sensor misreadings, erroneous data extractors, data conversions and even rounding errors. In other words, *precise* answers are often a myth: even when processing the entire data, the answer is still an approximate one.

Due to these reasons, a significant body of research in learning theory has accounted for this inherent noise when performing inference using training data. In fact, there has been much research on trade-off between inference error and computation [18]. Likewise, introducing a controlled degree of noise by database systems in exchange for significant savings in computational resources is quite natural.

2.4 Inference and Predictive Analytics are Probabilistic in Nature

The majority of the commonly used techniques for inference in machine learning, computational statistics, and stochastic modeling are based on probabilistic analysis. In some cases, the predictions themselves are probabilistic (e.g., in probabilistic regression [17]), while in other cases, the predictions are based on probabilistic reasoning. As these predictive models are designed to account for uncertainty and noise in their training data, adding a controlled degree of uncertainty to their input (e.g., by using a carefully selected sample) will not necessarily break their logic. In many cases, using a smaller sample of the data will lead to significant computational savings with no or little impact on the utility and accuracy of their predictions. This is because the time complexity of many learning algorithms is super-linear (e.g., cubic or quadratic) in their input size, while the improvement of their learning curve follows a diminishing return rule. In other words, the learning error is often proportional to $O(\frac{1}{\sqrt{n}})$ where n is the number of tuples used for training, e.g., to reduce the error from 20% to 2%, one typically needs to use $100\times$ more data (see Corollary 3.4 in [45]). In fact, in some cases, certain forms of data reduction can significantly improve accuracy, e.g., by sampling down the majority class and creating a more balanced distribution of different labels for classification tasks [40]. In general, sampling is a systematic way of reducing the data size while maintaining the essential properties of the data [43]. For instance, with proper sampling, the solution to the sub-sampled problem of linear regression is also a good approximate solution to the original problem [24].

Therefore, in many cases where the output of database queries are consumed by predictive analytics or machine learning algorithms, returning samples of the data or approximate results is a sound decision that can yield considerable performance benefits.

3 General Anatomy of Approximate Query Processing Systems

AQP systems can widely differ in their methodology and design choices. However, in this section, we attempt to provide a general anatomy by identifying the common architectural components of these systems. Our discussion of the various design decisions in each component will also serve as a natural taxonomy of the AQP systems. These components, shown in Figure 1, are discussed in the following subsections.

3.1 User Interface

This component is typically in charge of receiving queries and accuracy or latency requirements from the user, as well as communicating the approximate answers and the error bounds (or accuracy guarantees) back to the user. Some of these functionalities vary among different AQP systems. For example, systems providing an online aggregation [32, 65] interface do not take accuracy requirements from the user. Instead, they periodically report the current approximation and its estimated error as they are progressively processing the user query over a larger portion of the data. In these systems, the user can decide, based on the current accuracy of the running approximation (or the rate of its improvement), whether to terminate the query execution or await its completion over the entire data. As another exception, the user interface in certain AQP systems may not even return any error estimation to the user. These systems are often designed for debugging or real-time exploration purposes (e.g., in massive distributed systems [59] or scientific datasets [52]), and hence, the focus is on retrieving the outliers or corner cases. Here, an approximate answer—even without an accuracy guarantee—is often sufficient for a programmer or scientist to find a bug or suspect a trend in the underlying data. Once

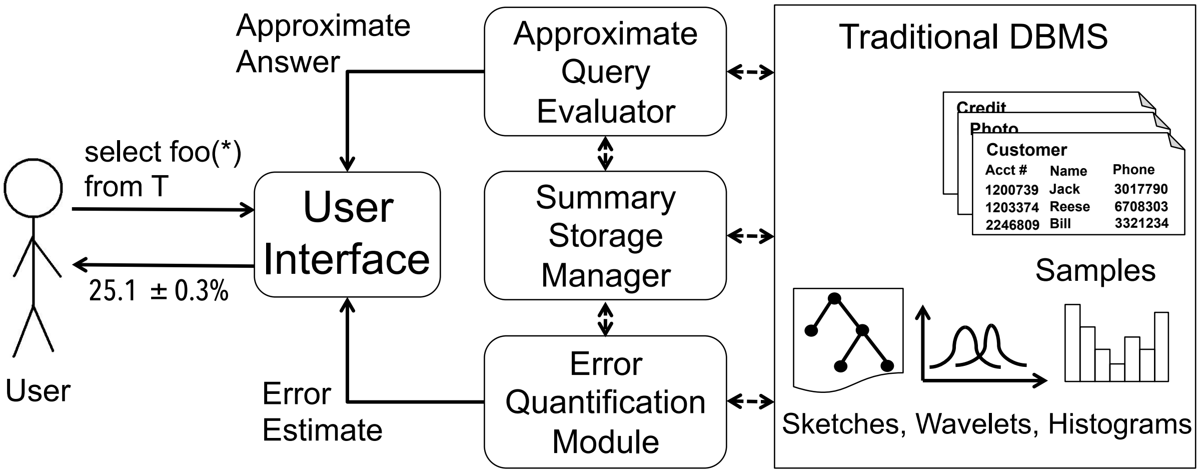


Figure 1: The general anatomy of an approximate query processing system.

a reasonable hypothesis is formed, an exact answer can be obtained by running on the entire data. However, most AQP systems allow the users to specify their accuracy or latency requirements (either on a per-query basis [12, 66] or as a system-wide setting), and report the accuracy of their results back to the user. The ability to specify and receive accuracy guarantees seems to an important factor in convincing users to use AQP systems. Min-max ranges [57], confidence intervals (a.k.a. error bars) [7, 16, 32, 66], additive errors [22], and probability of existence [23] have been used for communicating the error to the end users. However, given that the majority of database users are not statisticians, devising user-friendly and high-level mechanisms for expressing the error seems to be an area of research that requires more attention (see Section 7).

3.2 Summary Storage Manager

The summary storage manager is one of the key architectural components of an AQP system, responsible for (i) *the type of data summary* to use, (ii) *when to build* it, (iii) *how to store it*, and (iv) *when to update* it. The behavior, generality and efficiency of an AQP system depends heavily on the design choices made for how to handle these responsibilities in the summary storage manager.

Different types of data summaries — Various forms of data summaries and synopses have been utilized by AQP systems. *Samples* have been the most commonly used form of data summary. Several reasons have contributed to this popularity. By adhering to the same schema as the original table, samples support the widest range of queries compared to other types of data summaries. Besides their generality, samples are also extremely easy to implement. Most queries can be evaluated on a sample with little or no modifications to existing database engines. As one of the oldest concepts in statistics, there is also a large body of rich literature on the design and analysis of sampling-based approximations. Sampling also avoids the so called "curse of dimensionality"; the accuracy of a sample does not degrade with the the number of data dimensions. In particular, with random (a.k.a. uniform) sampling, the space required to store the sample only grows linearly with the number of attributes. Finally, most sampling algorithms do not use the AVI (Attribute Value Independence) assumption, and therefore, allow for high quality selectivity estimation [15]. Although virtually any query can be evaluated on a sample (rather than the entire dataset), the quality of the resulting approximation varies widely depending on the query. For example, evaluating `MIN`, `MAX`, top-K, and `COUNT DISTINCT` queries on a sample is often futile. Moreover, random samples are by definition inapt at capturing outliers in skewed distributions. As a result, many AQP systems have utilized various forms of *biased* samples in addition to random samples. In particular, *stratified sampling* has been a popular choice [8, 11, 13, 20, 35]. In survey design, stratified sampling has been traditionally used for reducing the overall variance by grouping similar items into the same stratum (and hence, reducing the intra-stratum variance). However, in AQP systems, stratified sampling has been used mainly for a different reason. Queries with highly selective `WHERE` clauses may match zero or only a few tuples in a random sample, leading to extremely large (or infinite) errors. By stratifying tuples based on the value of a subset of the attributes, any combination of those columns—no matter how rare—will be preserved in the sample [8, 11, 13, 20, 35]. Thus, a key challenge in using stratified samples is to decide which subsets of columns to stratify on. In other words, finding an optimal set of stratified samples is a physical design problem: given a fixed storage budget, choose subsets of columns that are most skewed, or are most likely to be filtered on by future queries (see [11] for a mathematical formulation of this problem). The choice of optimal set of stratified and random samples depends heavily on the specific queries that users want to run. However, since future queries are not always known *a priori* (especially, for exploratory and adhoc workloads [48]), special care needs to be taken to ensure that the selected samples remain optimal even when future workloads deviate from past ones (see [47] for a solution).

Histograms are another form of data summaries commonly used by AQP systems. A histogram is a binned representation of the (empirical) distribution of the data. One-dimensional (i.e., single-column) histograms are internally used by nearly all database systems to cost and selectivity estimation of different query plans. Since AQP systems are often built atop an existing DBMS, these histograms can also be used for producing fast approximate answers without incurring an additional storage overhead. The main drawback of histograms is that they suffer from the curse of dimensionality; the number of bins for k-column histograms grows exponentially with k. Using the AVI assumption to estimate the joint distribution of k columns from the 1-column histograms is equally futile (due to intra-column correlations).

Wavelet transforms are another means for summarizing large datasets, whereby a small set of wavelet coefficients can serve as a concise synopsis of the original data. While wavelets can exploit the structure of the data to provide considerable data compaction, they suffer from the same limitation as histograms, i.e., curse of dimensionality. In other words, one-dimensional wavelet synopses can only support a limited set of database queries. Moreover, the error guarantees traditionally offered by wavelet synopses are not always meaningful in an AQP context (see [30]).

There are numerous other types of synopses structures that can be built and used by AQP systems, including but not limited to sketches, coresets, and ϵ-nets. A detailed overview of various types of synopses is beyond the scope of this chapter.[3] While some of these data summaries are more practical than others, their common theme is that they are all specific to a certain type of query, but provide significantly better approximations than sampling-based techniques. Therefore, these types of synopses are typically used when users demand a specific query to be executed quickly and when there is an existing technique to support that type of query. Examples include HyperLogLog for `COUNT DISTINCT` queries [28], Count-Min-Sketch (CMS) for `COUNT-GROUP BY` queries [22, 42], Join Synopsis for foreign-key join queries [9], balanced box-decomposition (BBD) trees for range queries [14], and Neighbor Sensitive Hashing (KSH) tables for k-nearest neighbor queries [54].

The choice of the specific data summary depends on the target workload. AQP systems designed for speeding up a predetermined set of queries utilize various sketches, wavelets or histograms, whereas general-purpose AQP systems tend to rely on sampling strategies. Often, a combination of these data summaries are built to provide general support for most queries, while producing better approximations for those queries that are more important to the user. As a rule of thumb, building a random sample, several stratified samples, as well as a few common sketches is a reasonable choice.

Offline versus online data summarization — Another choice made by the Summary Storage Manager is whether to summarize the data prior to the arrival of the query (*offline*) or perform the summarization on demand after receiving the query (*online*). Performing the data summarization offline allows the AQP system to utilize expensive, more sophisticated types of synopses, e.g., summaries that require multiple passes over the data. At

[3] Interested readers are referred to [21] for an excellent and detailed discussion of different types of synopses.

the same time, the generated summaries can be further optimized through various physical layout strategies, such as replication, indexing, partitioning and caching (see below). However, there is a potential downside to the offline approach: the lack of precise knowledge of future queries can lead the Summary Storage Manager to making sub-optimal choices, e.g., by not building an appropriate stratified sample. In contrast, the online approach does not have to make any choices *a priori*. However, efficient data summarization in an online fashion is significantly more challenging. In other words, the online approach will incur an additional run-time overhead and will have fewer opportunities for building sophisticated types of synopses or auxiliary structures on demand.

Different storage strategies — The Summary Storage Manager is also in charge of the physical layout of the generated summaries. For example, this component may choose to store certain summaries at multiple granularities. For instance, it may physically store three samples of size $1\%, 2\%$ and 4%, respectively, to provide three logical samples of size $1\%, 3\%$ and 7%. The advantage of such logical samples, called *overlapping samples*, is that once the query is run on a 1% sample, its results on a 3% sample can be obtained by simply performing a delta computation using the 2% sample. This strategy is particularly useful when the initially chosen sample size does not meet the accuracy requirements of the user. An alternative strategy is to use each sample individually. Given the same storage budget, this choice provides a finer range of sample size selections, e.g., if the 1% size is deemed insufficient, the next choice will be 2% rather than 3%. However, delta computation will no longer be an option in this case. For most queries, delta computation is typically faster than a re-evaluation on the entire data. However, for non-monotonic or nested queries with `HAVING` clauses, this may not be the case [65].

Most AQP systems use an existing RDBMS to store their summaries, e.g., samples of a table are themselves stored as tables. However, the Summary Storage Manager can still decide which samples need to be cached in memory and which ones should be kept on disk [11]. For example, for time series data, the most recent window might be more likely to be queried, and hence, should be kept in memory. When on disk, the columnar or row-wise layout of a sample will affect its I/O cost as well as its ease-of-update. Whether and how to index each sample is yet another choice. In distributed AQP systems, partitioning and replication strategies applied to each sample will also have significant performance implications. For example, replicating a popular sample on every node may be a reasonable strategy unless it needs to be updated frequently.

Different maintenance strategies — There are several alternatives for the maintenance of the different synopses and summaries in an AQP system. In a *lazy* approach, the summaries are only updated as a batch, either periodically (e.g., once a week) or upon the user's explicit request. This approach is often preferred when the cost of updates is prohibitive, while the downside is the obvious staleness of the approximate results until the next update is performed. In an *eager* approach, the updates to the database are immediately reflected in the data summaries. For example, whenever a tuple is inserted to or deleted from a table, all samples on that table are also updated accordingly. Typically, data summaries are easier to update in face of insertions than deletions. As a result, most AQP systems only allow insert operations, which is sufficient for append-only environments, such as data streams or HDFS-based systems (e.g., Hive, Presto, Impala, SparkSQL).

3.3 Approximate Query Evaluator

This component is perhaps the heart of every AQP system, computing approximate answers for queries received through the User Interface. The overwhelming majority of the existing AQP systems are built atop an existing database system, for two main reasons: (i) to minimize the implementation effort as the majority of the logic from existing query processors can be rescued for approximation, and (ii) to provide users an option to always go back to the entire data and request a precise answer, if they decide to. The latter has been an important factor in improving the uptake of the AQP systems in the database market. In fact, one of the reasons behind the popularity of sampling-based approximations (compared to other forms of data summaries) is that they can be supported with minimal changes to the existing database systems.

The Approximate Query Evaluator has to decide whether the given query is of a form that can be answered with any of the available data summaries. Adhoc synopses (e.g., sketches) are often preferred over more generic types of synopses. Otherwise, the system has to decide if and which of the existing samples can be used to best meet the user's requirement. When applicable to the query, stratified samples are preferred over random samples. Once a sample type is decided, an appropriate fraction of it is chosen based on the latency and error requirements.

Once an appropriate data summary is selected, the Approximate Query Evaluator either directly evaluates the query using that structure, or, in most cases, issues certain queries to the underlying database to evaluate the query on its behalf. When possible, the latter approach allows the AQP system to take full advantage of the mature query optimizers and the efficient implementation of off-the-shelf database systems. Certain post-processing steps, such as scaling and bias correction, may also be applied to the results before returning an approximate answer to the user.

To verify whether the user's accuracy requirements are met, the Approximate Query Evaluator needs to communicate with the Error Quantification Module, either during the query evaluation or at the end of the process. While a more sophisticated Approximate Query Evaluator can perform multi-query optimization, resource management, or load balancing, in practice most AQP systems simply rely on an underlying off-the-shelf query engine to perform these tasks for them.

3.4 Error Quantification Module

An approximate answer is as useful as its accuracy guarantees. The Error Quantification Module (EQM) is in charge of quantifying or estimating the extent of error in the approximate results. Some EQMs provide an error estimate only for the final answer [7, 11], while others provide a running error estimate as the query is progressing (i.e., online aggregation [32, 65]). The latter allow users to terminate the execution as soon as they are satisfied with the current estimate or otherwise decide to abandon their current query. However, there are few AQP systems that do not provide any error guarantees, and simply return a best-effort approximation (see [52, 59]).

Another important distinction between various error quantification strategies is their ability to *predict the error* without evaluating the query. For example, given a sample of the data, a closed-form error can predict the error of an `AVG` query using a few basic statistics, e.g., the sample size and the estimated variance of the data. On the other hand, more sophisticated error estimation strategies (e.g., simulation-based bootstrap) will only know the error once they perform an expensive computation, i.e., after evaluating the query on the sample. The ability to predict the approximation is an important factor, as it allows the AQP system to choose the best type and resolution among the available data summaries according to the latency or accuracy goals of the user. When this ability is missing, the EQM module has to resort to trial-and-error: the EQM will have to use heuristics to choose a data summary, evaluate the query on that summary, and then quantify the error to see if the results meet the accuracy requirements (otherwise, the query will be repeated on a different data summary). There are different forms of approximation error used in AQP systems, as described next.

Variance — Many AQP systems have used variance as a measure of quality for their sampling-based approximate answers. Other types of data summaries have rarely been analyzed for their variance. This is perhaps due to the large body of statistical literature on the variance of sampling and survey estimators. The work on variance estimation in the context of AQP systems can be categorized into two main approaches. The first approach [11, 19, 20, 32, 34, 25, 36, 53, 64] analytically derives closed-form error estimates for simple aggregate functions, such as `SUM`, `AVG` and `COUNT`. Although computationally appealing, analytic error quantification suffers from its lack of generality. For every new type of queries, a new formula must be derived. This derivation is a manual process that is ad-hoc and often impractical for complex queries [56].

A second approach to error estimation is to use the so called *bootstrap*. By providing a routine method for estimating errors, bootstrap is a far more general method than closed forms [10, 39, 56]. The downside of bootstrap is its computational overhead, as hundreds or thousands of bootstrap trials are typically needed to obtain reliable estimates. However, several optimizations have been proposed for speeding up the bootstrap process (see Section 6.2).

Assuming asymptotic normality (and an unbiased answer), the variance is sufficient for computing a confidence interval for the approximate answer. Rather than reporting the variance, a confidence interval is often used to communicate the error to the end user.

Bias — Bias is another important measure of quality for approximate answers. We would like to minimize the bias in our approximations, or ideally, have unbiased answers. While the formal definition of bias can be found in Section 4, its intuitive meaning is quite simple: an unbiased approximation is one where the chances of an over-estimation is the same as that of an under-estimation. In other words, if the approximation procedure is repeated many times, the results will be centered around the true answer. While taken seriously in the statistics literature, the bias is often overlooked in Error Quantification Modules of AQP systems. Most AQP systems assume that the bias approaches zero as the sample size increases. We will show, in Section 6, that this assumption is not necessarily true.

Probability of existence — Bias and variance are used for assessing the approximation quality of a numerical estimate. However, the output of database queries is not a single number, but a relation. Due to the propagation of various forms of uncertainty during query processing, certain tuples may appear in the output of the approximate answer that will not appear in the output of the exact query (see Section 5). To assess the likelihood of this phenomenon, the EQM computes a different form of error, called *the probability of existence*. The probability of existence of an output tuple is the probability with which it would also appear in the output of the exact query. This form of error is well-studied in the context of probabilistic databases [23]. In an EQM, these probabilities can be easily computed using bootstrap [67].

Other forms of error — There are many other (less common) forms of error used by EQMs for various types of data summaries. For example, *coresets*—a carefully selected subset of the original tuples used for approximating computational geometry problems, such as as k-medians and k-means— provide multiplicative error guarantees of $1 \pm \epsilon$ [31].. In other words, coresets guarantee that $(1-\epsilon)\theta \leq \hat{\theta} \leq (1+\epsilon)$, where θ is the true answer, $\hat{\theta}$ is the approximate answer, and $\epsilon > 0$ is a pre-specified parameter chosen by the user (the smaller the ϵ, the more accurate the results but the larger the coreset size). For ϵ-nets (another subset selection approximation for geometric problems), the guarantees state that any range query containing at least $\epsilon > 0$ fraction of the original tuples will also contain at least one tuple of the selected subset. The quality of wavelet-based reconstructions is often expressed in terms of the L_2-norm error of the approximation. However, there are thresholding algorithms for Haar wavelets that minimize other forms of error, such as the maximum absolute or relative error of the approximate results (see [21] and the references within). Other AQP systems have also used absolute range guarantees [57] as a measure of accuracy. Typically, none of these error guarantees are easily convertible into one another. Consequently, most AQP systems do not combine multiple types of data summaries for answering the same query, i.e., the entire query is evaluated only using one type of data summary.

4 Not All Samples are Created Equal

Let D be our dataset (e.g., a large table), q be a query, and $q(D)$ be the result of evaluating q on the entire dataset D. In sampling-based AQP systems, the goal is to use a small sample of D, say S, to provide an approximate answer to q. This is typically achieved by running a (potentially) modified version of q, say q', on S, such that $q'(S) \approx q(D)$. In this case, q' is called an estimator for q. When $|S| << |D|$, $q'(S)$ can often be computed much more efficiently than $q(D)$. An important characteristic of an estimator is its *bias* b, defined as follows:

$$b = \mathbb{E}_{|S|=n}[q'(S)] - q(D) \tag{1}$$

Since the expectation is always over all possible samples of size n, henceforth we omit the $|S| = n$ subscript from our expectation notation. Also, for ease of presentation, we assume that queries only return a single number (though these definitions can be easily extended to relations). An ideal estimator is one that is *unbiased*, namely:

$$b = \mathbb{E}_{|S|=k}[q'(S)] - q(D) = 0$$

Unfortunately, in general, deriving an unbiased estimator q' can be extremely difficult and adhoc. As a result, most AQP systems use a heuristic, which we call the *plug-in approach* and describe in Section 4.1. (However, we later show (in Section 6) that this plug-in approach does not guarantee an unbiased estimate and thus, an additional bias correction step is often needed.) After a brief overview of the most common sampling strategies (Section 4.2), we provide a list of estimators—along with their bias and variance closed forms—for simple aggregate functions under each strategy (Section 4.3). In particular, to the best of our knowledge, some of our closed-form estimates for conditional aggregates (i.e., aggregates with a WHERE clause) have not been previously derived in the literature. We list these new and existing closed-form estimates in concise tables to serve as an accessible reference for non-statisticians (e.g., practitioners and engineers) involved in implementing AQP systems.

4.1 Plug-in Approach

To provide reasonable approximations without having to modify every query or make significant changes to existing query evaluation engines, the following heuristic is used by almost all AQP systems:

1. Assign an appropriate weight w_i to each sampled tuple t_i according to its sampling rate. For example, in the original table (before sampling) the weight of every tuple is 1, whereas in a random sample comprised of 1% of the original tuples $w_i = 0.01$ for all t_i. Different tuples can have different weights, e.g., in stratified sampling (see Section 4.2) the tuples in each stratum have a different sampling rate, and hence, a different weight.

2. Modify the aggregate functions (e.g., SUM, AVG, COUNT, quantiles) to take these weights into account in their computation. For example, to calculate the SUM of attribute A in n tuples, use $\sum_{i=1}^{n} \frac{t_i \cdot A}{w_i}$ (instead of $\sum_{i=1}^{n} t_i \cdot A$), where $t_i \cdot A$ is the value of attribute A in tuple t_i. Similarly, the AVG of attribute A will be computed as $\frac{1}{n}\sum_{i=1}^{n} \frac{A_i}{w_i}$ (instead of $\frac{1}{n}\sum_{i=1}^{n} A_i$). COUNT and quantile aggregates can be modified similarly.[4] (This scaling is also known as the Horvitz-Thompson (HT) estimator [33].)

3. After the two modifications above, simply run the same query (i.e., $q' = q$) on sample S to produce an approximation of $q(D)$, namely:
$$q(D) \approx q(S) \tag{2}$$

We call this the plug-in approach since we simply plug in a sample instead of the original data but run the same query. While this approach might seem intuitive and natural, it does not necessarily produce unbiased answers (see Section 6). Next, we briefly enumerate the most commonly used sampling strategies in AQP systems.

[4] While similar weighting could also be used for MIN and MAX, the resulting estimators would still be biased for these extreme statistics.

4.2 Sampling Strategies

Given the popularity of sampling techniques in AQP systems, in this section we briefly review the most common sampling strategies employed in these systems. Subsequently, in Section 4.3, we present the closed form expressions of the error for each strategy. We refer the reader to [61] for a detailed comparison of different sampling strategies, [62] for different sampling algorithms, and [29] for a discussion of updating samples.

Fixed-size sampling with replacement — Given a fixed sample size n, we choose n tuples from D, independently and at random, and include them in our sample S. Each item can be sampled multiple times. Throughout this chapter, we use sets and multisets interchangeably, i.e., we allow duplicate values in a set. Thus, even though there might be duplicate tuples in S, we still have $|S| = n$. This form of sampling allows for the use of bootstrap, which is a powerful error estimation technique (see Section 6.2). However, it is rarely used in AQP systems since, given the same sample size, sampling without replacement can capture more information about the underlying data.

Fixed-size sampling without replacement — Given a fixed sample size n, we choose n tuples from D at random, and include them in our sample S. Each tuple can be sampled at most once. When $n << |D|$, sampling with replacement behaves similarly to sampling without replacement. As a result, even thought bootstrap is only designed for the former, in practice it is also used for the latter when this condition holds. When maintained for a streaming dataset (where items can be added or removed), fixed-size sampling without replacement is referred to as *reservoir sampling* [62]. Probabilistic variations of this form of sampling are also available, whereby tolerating a small possibility of error (in producing the requested sample size) allows for an efficient and embarrassingly parallel implementation [43].

Bernoulli sampling without scaling — Given a sampling ratio P ($0{<}P{\leq}1$), draw a random number r in $[0, 1]$ for every tuple T in D; include T in S if $r \leq P$. This is the classical formulation of Bernoulli sampling, which is rarely used in AQP systems (see below).

Bernoulli sampling with scaling — Given a sampling ratio P ($0{\leq}P{\leq}1$), draw a random number r in $(0, 1]$ for every tuple T in D; include $\frac{1}{P}T$ in S if $r \leq P$, and otherwise add a **0** to S. Here, $\frac{1}{P}T$ is tuple T with all its attributes multiplied by $\frac{1}{P}$, and **0** is a tuple with 0 for all attributes. Thus, the sample size is the same as the original data, i.e., $|S| = |D| = N$. Despite this conceptually large sample size, this sampling technique has the same performance advantage as Bernoulli sampling without scaling since the **0** tuples are not necessarily stored and evaluated during query evaluation. The main advantage of Bernoulli sampling with scaling over Bernoulli sampling without scaling is that the sample size is no longer a random variable [51]. This greatly simplifies the derivation of the closed form error estimation (see Table 2). This type of sampling can be easily implemented for streaming or distributed environments (e.g., load shedding in a data stream management system [60]).

Stratified sampling — Given L strata $D_1, \cdots, D_L$, sample S_i of size n_i is chosen from each stratum D_i *independently*, using fixed-size sampling without replacement.[5] Our sample S is defined as $S = \cup_{i=1}^{L} S_i$, and we have $n = |S| = \cup_{i=1}^{L} n_i$.

The main advantage of stratified sampling is the potential reduction of variance, especially when the tuples are similar within each stratum but dissimilar across different strata. In AQP systems, stratified sampling has another (even more important) advantage. If the tuples are stratified based on the value of their attributes with a skewed distribution, then queries will be guaranteed to match at least a minimum number of tuples, even if their filtering conditions match rare values of those columns. This is different than classical survey sampling. In an AQP system, the effective population changes depending on the filtering conditions of the query (see Table 2 in Section 4.3).

While various sampling strategies differ in their ease of updates and amenability to parallelism, they also

[5] In this chapter, we only consider fixed-size sampling without replacement per stratum. However, combining the error of multiple strata follows the same principle regardless of the sampling strategy applied to each stratum.

widely differ in their approximation error and ease of closed-form estimation. In the next section, we will present the closed-form estimations of bias and variance for simple aggregate queries under each of these sampling strategies.

4.3 Closed-form Error

Our main goal in this section is to provide practitioners (and non-statisticians) a concise and accessible list of closed-form estimators for common forms of aggregates and sampling strategies.[6]

The sample estimators are widely discussed in the statistics literature. However, deriving and applying such estimators in an AQP context requires additional care. This is because database queries often have a `WHERE` clause, which effectively changes the population D and the sample S for every new query. Thus, the population size of tuples satisfying condition c can no longer be known in advance. Although the fraction of tuples in D satisfying c, F_c, can be computed by evaluating all tuples in D, such an approach is too costly. To enable fast computation, only a sample S of D must be accessed during query evaluation. For example, the fraction of tuples in S satisfying c, f_c, can be used as an approximation of F_c. However, f_c, will no longer be a constant but a random variable, and its variance must be taken into account when estimating the variance of aggregate queries. Even worse, when $f_c = 0$, one must still consider the possibility that $F_c \neq 0$. These challenges translate to more complicated closed-forms for aggregate queries with a `WHERE` clause.

In the following discussion, we divide different statistics into *computable* and *incomputable* to distinguish them based on whether they can be computed efficiently. We call a statistic or estimator *computable* if it can be computed from the original population without knowing the condition c in the query's `WHERE` clause, or from the sample. For example, N, n, and $\sum_{i=1}^{n} I_c(Y_i)$ are all computable statistics. Although the latter depends on c, it can be computed from the sample and hence, is computable. Likewise, $\gamma_c = \frac{N}{n}\sum_{i=1}^{n} I_c(Y_i)$ is also a computable estimator, as it only requires computable statistics. In contrast, we call a statistic or estimator *incomputable* if it requires certain statistics of the original population that depend on the query's `WHERE` clause condition c, or from the sample. For example, $\bar{X}_c$ and σ_{D_c} are incomputable statistics.

For ease of reference, we have summarized all our notation in Table 1. In this table, the target attribute refers to the attribute being aggregated on by the query. Table 2 provides a concise list of various estimators for simple aggregate queries under different sampling strategies.[7] The results of Table 2 can be easily generalized to `SELECT` clauses with multiple aggregates or aggregates over arithmetic expressions of the original attributes. Although omitted from Table 2, supporting the `GROUP BY` clause is also straightforward. Conceptually, each query can be re-evaluated once per every group g, each time with a new predicate c_g added its `WHERE` clause, such that c_g evaluates to true only for tuples in group g. Finally, the condition c in Table 2 is any condition that can be evaluated for each tuple in D without accessing other tuples in D. In other words, c can be either a simple condition on various attributes of tuple $T \in D$, or a nested query that does not reference D.

In Table 2, for each pair of aggregate and sampling type, we have provided the following:

1. A computable (and typically unbiased) *query estimator* for computing the approximate answer. For example, to approximate a conditional count query under fixed-sized sampling without replacement, γ_c can be computed using only the number of tuples in S satisfying c (i.e., $\sum_{i=1}^{n} I_c(Y_i)$, the size of the sample n, and the total size of the original population N.

2. The *exact expression of the query estimator's standard deviation*. However, this expression is often incomputable.

3. A computable *unbiased estimator for the query estimator's variance*, when possible. This estimator is important because the exact expression of the query estimator's standard deviation (e.g., σ_{γ_c}) is incom-

[6] The error estimation of more complex queries requires (analytical or Monte-Carlo) bootstrap, which will be discussed in Section 6.2.

[7] The references, proofs and derivations of these closed-form estimations are deferred to our technical report [49].

putable (see above).[8] When such an estimator is not available, we have instead provided computable unbiased estimators (under the right-most column of Table 2) for each of the incomputable statistics used in the expression of the query estimator's standard deviation. By substituting such statistics with their computable unbiased estimators, one can compute a reasonable (but biased) approximation of the query estimator's standard deviation.

To use the formulas presented in this chapter, one must use the following definitions of population and sample standard deviation:[9].

$$\sigma_D = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})^2} \text{ and } \sigma_S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

While in this section we focused on traditional error metrics (i.e., bias and variance), in the next section we discuss other (subtle) forms of error that can occur in an AQP system.

5 Subtle Types of Error: Subset and Superset Errors

Consider a IRS (Internal Revenue Service) table storing the income and age of different citizens living in different parts of the country:

```
IRS (citizen_name, city, age, income)
```

Now consider the following query against this table:

```
SELECT city, avg(income) as avg_income
FROM IRS
GROUP BY city
```

Let us assume that the true answer (i.e., the answer of running this query against the original table) is as follows.

Now assume that the query above is evaluated against only a sample of the original table. Then, the output aggregate values will be approximate. For example:

Here ± represents a confidence interval at a certain level of confidence, e.g., 95%. The problem here is that if a city (e.g., Ann Arbor) has significantly fewer tuples in the original table than larger cities (e.g., New York), then it may not be present in the selected sample. Thus, using such a sample will naturally leave out the group pertaining to Ann Arbor from the query's output altogether. When this situation occurs, the error is no longer one of *aggregation error*. In other words, Ann Arbor will not be even present in the output to be accompanied with accuracy guarantees for its aggregate value (here, average income). This phenomenon has been the motivation behind the wide use of stratified samples, increasing the likelihood of rare groups being present in the sample [8, 20, 11, 66]. Even when a reliable estimate cannot be obtained for a particular group, one would wish to at least know that such a group should have existed. In other words, the following would be an ideal output:

Unfortunately, the use of stratified samples does not always prevent missing records from the output. This problem, called *subset error*, becomes more evident by considering the following slightly more involved query.

[8] The square root of our unbiased estimators for variance may be used as an approximation of the standard deviation. However, note that the square root of an unbiased estimator for a random variable is not necessarily an unbiased estimator for the square root of that random variable!

[9] In other words, we do *not* use the $\sigma_S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$ definition. Instead, we apply a correction factor separately.

Symbol	**Meaning**
D	The entire set of tuples
S	The selected sample of tuples, i.e., $S \subseteq D$
N	The number of original tuples, i.e., $N = \lvert D \rvert$
n	The number of tuples in the sample, i.e., $n = \lvert S \rvert$
c	The condition used in the `WHERE` clause
I_c	The indicator function, where $I_c(t) = 1$ if tuple t satisfies c and $I_c(t) = 0$ otherwise
F_c	The fraction of tuples in D satisfying condition c $(0 \le F_c \le 1)$
f_c	The fraction of tuples in S satisfying condition c $(0 \le f_c \le 1)$
N_c	The number of tuples in D satisfying condition c $(0 \le N_c \le N)$
$T_1, \cdots, T_N$	The tuples in D
$t_1, \cdots, t_n$	The tuples in S
$X_1, \cdots, X_N$	The values of the target attribute in $T_1, \cdots, T_N$
$Y_1, \cdots, Y_n$	The values of the target attribute in $t_1, \cdots, t_n$
$Z_{c_1}, \cdots, Z_{c_N}$	$Z_{c_i} = X_i$ if $I_c(T_i) = 1$ and $Z_{c_i} = 0$ otherwise
$\bar{X}$	The mean of the target attribute in D, i.e., $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$
$\bar{Y}$	The mean of the target attribute in S, i.e., $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$
$\bar{X}_c$	The conditional mean of the target attribute in D, i.e., $\bar{X}_c = \frac{\sum_{i=1}^{N} X_i I_c(X_i)}{\sum_{i=1}^{N} I_c(X_i)}$
$\bar{Y}_c$	The conditional mean of the target attribute in S, i.e., $\bar{Y}_c = \frac{\sum_{i=1}^{n} Y_i I_c(Y_i)}{\sum_{i=1}^{n} I_c(Y_i)}$
$\bar{Z}_c$	The mean of the Z_{c_i} values, i.e., $\bar{Z}_c = \frac{1}{N}\sum_{i=1}^{N} Z_{c_i}$
σ_D	The standard deviation of the target attribute in D, i.e., $\sigma_D = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})^2}$
σ_S	The standard deviation of the target attribute in S, i.e., $\sigma_S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$
σ_Z	The standard deviation of the Z_{c_i} values for tuples in D, i.e., $\sigma_Z = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Z_{c_i} - \bar{Z}_c)^2}$
σ_{ZY}	The standard deviation of the Z_{c_i} values for tuples in S
σ_{D_c}	The standard deviation of the target attribute in $\{X_i \mid T_i \in D,\ I_c(T_i) = 1\}$
σ_{S_c}	The standard deviation of the target attribute in $\{Y_i \mid t_i \in S,\ I_c(t_i) = 1\}$
P	The sampling probability in Bernoulli sampling $(0 < P \le 1)$
L	The number of strata in stratified sampling $(1 \le L \le N)$.
$D_1, \cdots, D_L$	The strata of D, i.e., $D = \cup_{i=1}^{L} D_i$ and $D_i \cap D_j = \emptyset$ for $i \neq j$
$S_1, \cdots, S_L$	The samples from different strata of D, i.e., $S = \cup_{i=1}^{L} S_i$ and $S_i \subseteq D_i$
$N_1, \cdots, N_L$	The number of tuples in $D_1, \cdots, D_L$, i.e., $N_i = \lvert D_i \rvert$ and $N = \sum_{i=1}^{L} N_i$
$N_{c_1}, \cdots, N_{c_L}$	The number of tuples in $D_1, \cdots, D_L$ satisfying c $(0 \le N_{c_i} \le N_i)$
$n_1, \cdots, n_L$	The number of tuples in $S_1, \cdots, S_L$, i.e., $n_i = \lvert S_i \rvert$ and $n = \sum_{i=1}^{L} n_i$
$\bar{X}_1, \cdots, \bar{X}_L$	The mean of the target attribute in $D_1, \cdots, D_L$
$\bar{Y}_1, \cdots, \bar{Y}_L$	The mean of the target attribute in $S_1, \cdots, S_L$
$F_{c_1}, \cdots, F_{c_L}$	The fraction of tuples satisfying c in $D_1, \cdots, D_L$
$f_{c_1}, \cdots, f_{c_L}$	The fraction of tuples satisfying c in $S_1, \cdots, S_L$
σ_{D_i}	The standard deviation of the target attribute in D_i
σ_{S_i}	The standard deviation of the target attribute in S_i
$\sigma_{D_{c_i}}$	The standard deviation of the target attribute for tuples in D_i that satisfy c
$\sigma_{S_{c_i}}$	The standard deviation of the target attribute for tuples in S_i that satisfy c
σ_{Z_i}	The standard deviation of the Z_{c_i} values for tuples in D_i
σ_{ZY_i}	The standard deviation of the Z_{c_i} values for tuples in S_i
θ	The estimator for approximating $\bar{X}$ using S ($\texttt{AVG}(X)$ $\texttt{FROM}$ D)
γ_c	The estimator for approximating $\sum_{i=1}^{N} I_c(X_i)$ using S ($\texttt{COUNT} *$ $\texttt{FROM}$ D $\texttt{WHERE}$ c)
θ_c	The estimator for approximating $\bar{X}_c$ using S ($\texttt{AVG}(X)$ $\texttt{FROM}$ D $\texttt{WHERE}$ c)
ϕ_c	The estimator for approximating $\sum_{i=1}^{N} X_i I_c(X_i)$ using S ($\texttt{SUM}(X)$ $\texttt{FROM}$ D $\texttt{WHERE}$ c)
σ_θ	The standard deviation of θ
σ_{γ_c}	The standard deviation of γ_c
σ_{θ_c}	The standard deviation of θ_c
σ_{ϕ_c}	The standard deviation of ϕ_c

Table 1: Notation used in this paper.

Sampling **Strategy**	**Mean Queries:** `SELECT AVG(X) FROM D`	**Conditional Mean Queries:** `SELECT AVG(X) FROM D WHERE c`	**Conditional Sum Queries:** `SELECT SUM(X) FROM D WHERE c`	**Conditional Count Queries:** `SELECT COUNT(*) FROM D WHERE c`	**Other sample-based estimators** **for population-based quantities**
Fixed-size with Replacement	$\theta = \bar{Y}$ (unbiased) $\sigma_\theta = \frac{\sigma_D}{\sqrt{n}}$ $\sigma^2_\theta = \mathbb{E}\left[\frac{1}{n-1}\sigma^2_S\right]$	$\theta_c = \bar{Y}_c$ (biased: $\mathbb{E}\left[\theta_c\right] = (1-(1-F_c)^n)\bar{X}_c$) $\sigma_{\theta_c} = \sqrt{\sum_{k=1}^{n} \frac{\sigma^2_{D_c}}{k}\binom{n}{k}F_c^k(1-F_c)^{n-k} + \bar{X}_c^2(1-(1-F_c)^n)(1-F_c)^n}$ (no computable unbiased estimator for $\sigma^2_{\theta_c}$ unless $F_c = 1$)	$\phi_c = \frac{N}{n}\sum_{i=1}^{n} Y_i I_c(Y_i)$ (unbiased) $\sigma_{\phi_c}{=}N\frac{\sigma_Z}{\sqrt{n}}$ $\sigma^2_{\phi_c} = \mathbb{E}\left[\frac{N^2}{n-1}\sigma^2_{ZY}\right]$	$\gamma_c = N f_c$(unbiased) $\sigma_{\gamma_c} = N\sqrt{\frac{F_c(1-F_c)}{n}}$ $\sigma^2_{\gamma_c} = \mathbb{E}\left[\frac{N^2}{n-1}f_c(1-f_c)\right]$	$\sigma^2_{D_c} = \mathbb{E}\left[\frac{1}{M}\sigma^2_{S_c}\right]$ $M = \sum_{k=1}^{n}\frac{k-1}{k}\binom{n}{k}F_c^k(1-F_c)^{n-k}$ $F_c = \mathbb{E}\left[f_c\right]$
Fixed-size without Replacement	$\theta = \bar{Y}$ (unbiased) $\sigma_\theta = \sigma_D\sqrt{\frac{N-n}{n(N-1)}}$ $\sigma^2_\theta = \mathbb{E}\left[\frac{N-n}{N(n-1)}\sigma^2_S\right]$	$\theta_c = \bar{Y}_c$ $\mathbb{E}\left[\theta_c\right] = \bar{X}_c W \begin{cases} biased & if\ n \leqslant N - N_c \\ unbiased & if\ n > N - N_c \end{cases}$ $\sigma_{\theta_c} = \sqrt{\sigma^2_{D_c}\frac{N_c}{N_c-1}H_c + \bar{X}_c^2(1-W)W}$ $a = max\{1, n-N+N_c\},\ b = min\{n, N_c\},$ $W = \begin{cases} \sum_{k=a}^{b}\frac{\binom{N_c}{k}\binom{N-N_c}{n-k}}{\binom{N}{n}} & n \leqslant N - N_c \\ 1 & n > N - N_c \end{cases}$ $H_c = \sum_{k=a}^{b}\left(\frac{1}{k} - \frac{1}{N_c}\right)\frac{\binom{N_c}{k}\binom{N-N_c}{n-k}}{\binom{N}{n}}$ (no computable unbiased estimator for $\sigma^2_{\theta_c}$ unless $F_c = 1$)	$\phi_c = \frac{N}{n}\sum_{i=1}^{n} Y_i I_c(Y_i)$ (unbiased) $\sigma_{\phi_c}{=}N\sigma_Z\sqrt{\frac{N-n}{n(N-1)}}$ $\sigma^2_{\phi_c} = \mathbb{E}\left[\frac{N(N-n)}{n-1}\sigma^2_{ZY}\right]$	$\gamma_c = N f_c$ (unbiased) $\sigma_{\gamma_c} = N\sqrt{\frac{N-n}{n(N-1)}F_c(1-F_c)}$ $\sigma^2_{\gamma_c} = \mathbb{E}\left[\frac{N(N-n)}{n-1}f_c(1-f_c)\right]$	$\sigma^2_{D_c} = \mathbb{E}\left[\frac{1}{M}\sigma^2_{S_c}\right]$ $M = \frac{N_c}{N_c-1}\sum_{k=a}^{b}\frac{k-1}{k}\frac{\binom{N_c}{k}\binom{N-N_c}{n-k}}{\binom{N}{n}}$ $F_c = \mathbb{E}\left[f_c\right]$ $N_c = \mathbb{E}\left[N f_c\right]$ $a = max\{1, n-N+N_c\},\ b = min\{n, N_c\}$
Bernoulli without Scaling	$\theta = \bar{Y}$ (biased: $E[\theta] = (1-(1-P)^N)\bar{X}$) $\sigma_\theta = \sqrt{\frac{N}{N-1}\sigma^2_D H + (1-P)^N(1-(1-P)^N)\bar{X}^2}$ $H{=}\sum_{k=1}^{N}(\frac{1}{k})\binom{N}{k}P^k(1-P)^{N-k}{-}\frac{(1-(1-P)^N)}{N}$	$\theta_c = \bar{Y}_c$ (biased: $E[\theta_c] = (1-(1-P)^{N_c})\bar{X}_c$) $\sigma_{\theta_c} = \sqrt{\frac{N_c}{N_c-1}\sigma^2_{D_c}H_c + (1-P)^{N_c}(1-(1-P)^{N_c})\bar{X}_c^2}$ $H_c{=}\sum_{k=1}^{N_c}(\frac{1}{k})\binom{N_c}{k}P^k(1-P)^{N_c-k}{-}\frac{(1-(1-P)^{N_c})}{N_c}$	$\phi_c = \frac{N}{n}\sum_{i=1}^{n} Y_i I_c(Y_i)$ (biased: $\mathbb{E}\left[\phi_c\right] = (1-(1-P)^N)N\bar{Z}$) $\sigma_{\phi_c}{=}N\sqrt{\frac{N}{N-1}\sigma^2_Z H + (1-P)^N(1-(1-P)^N)\bar{Z}^2}$ $H = \sum_{k=1}^{N}(\frac{1}{k})\binom{N}{k}P^k(1-P)^{N-k} - \frac{(1-(1-P)^N)}{N}$	$\gamma_c = N f_c$ (biased: $\mathbb{E}\left[\gamma_c\right] = (1-(1-P)^N)N_c$) $\sigma_{\gamma_c} = N\sqrt{F_c(1-F_c)\frac{N}{N-1}H + (1-P)^N(1-(1-P)^N)F_c^2}$ $H = \sum_{k=1}^{N}(\frac{1}{k})\binom{N}{k}P^k(1-P)^{N-k} - \frac{(1-(1-P)^N)}{N}$	$\sigma^2_{D_c} = \mathbb{E}\left[\frac{1}{M}\sigma^2_{S_c}\right]$ $M = \sum_{k=1}^{N_c}\frac{N_c(k-1)}{k(N_c-1)}\binom{N_c}{k}P^k(1-P)^{N_c-k}$ $F_c = \mathbb{E}\left[\frac{f_c}{1-(1-P)^N}\right]$ $N_c = \mathbb{E}\left[\frac{N f_c}{1-(1-P)^N}\right]$
Bernoulli with Scaling	$\theta = \frac{1}{N}\sum_{i=1}^{n} Y_i/P$ (unbiased) $\sigma_\theta = \sqrt{\frac{1-P}{NP}(\sigma^2_D + \bar{X}^2)}$ $\sigma^2_\theta = \mathbb{E}\left[\frac{1-P}{N^2P^2}\sum_{i=1}^{n} Y_i^2\right]$	$\theta_c = \frac{\sum_{i=1}^{n} Y_i I_c(Y_i)/P}{\sum_{i=1}^{n} I_c(Y_i)/P} = \bar{Y}_c$ (biased: $E[\theta_c] = (1-(1-P)^{N_c})\bar{X}_c$) $\sigma_{\theta_c} = \sqrt{\frac{N_c}{N_c-1}\sigma^2_{D_c}H_c + (1-P)^{N_c}(1-(1-P)^{N_c})\bar{X}_c^2}$ $H_c{=}\sum_{k=1}^{N_c}(\frac{1}{k})\binom{N_c}{k}P^k(1-P)^{N_c-k}{-}\frac{(1-(1-P)^{N_c})}{N_c}$	$\phi_c = \sum_{i=1}^{n} Y_i I_c(Y_i)/P$ (unbiased) $\sigma_{\phi_c}{=}\sqrt{\frac{1-P}{P}N(\sigma^2_Z + \bar{Z}^2)}$ $\sigma^2_{\phi_c} = \mathbb{E}\left[\frac{1-P}{P^2}\sum_{i=1}^{n} Y_i^2 I_c(Y_i)\right]$	$\gamma_c = \sum_{i=1}^{n} I_c(Y_i)/P$ (unbiased) $\sigma_{\gamma_c} = \sqrt{\frac{1-P}{P}NF_c}$ $\sigma^2_{\gamma_c} = \mathbb{E}\left[\frac{1-P}{P^2}\sum_{i=1}^{n} I_c(Y_i)\right]$	$\sigma^2_{D_c} = \mathbb{E}\left[\frac{1}{M}\sigma^2_{S_c}\right]$ $M = \sum_{k=1}^{N_c}\frac{N_c(k-1)}{k(N_c-1)}\binom{N_c}{k}P^k(1-P)^{N_c-k}$
Stratified Sampling	$\theta = \frac{1}{N}\sum_{i=1}^{L} N_i\bar{Y}_i$ (unbiased) $\sigma_\theta = \frac{1}{N}\sqrt{\sum_{i=1}^{L}\frac{(N_i-n_i)}{n_i(N_i-1)}N_i^2\sigma^2_{Di}}$ $\sigma^2_\theta = \mathbb{E}\left[\frac{1}{N^2}\sum_{i=1}^{L}\frac{N_i(N_i-n_i)}{n_i-1}\sigma^2_{S_i}\right]$	$\theta_c = \frac{1}{N}\sum_{i=1}^{L} N_i\frac{\sum_{Y_j\in S_i} Y_j I_c(Y_j)}{\sum_{Y_j\in S_i} I_c(Y_j)}$ $\mathbb{E}\left[\theta_c\right] = \frac{1}{N}\sum_{i=1}^{L} N_i\bar{X}_{c_i}W_i \begin{cases} unbiased & if\ n_i > N_i - N_{c_i} \\ & for\ all\ i \in \{1, \cdots, L\} \\ biased & otherwise \end{cases}$ $\sigma_{\theta_c} = \frac{1}{N}\sqrt{\sum_{i=1}^{L} N_i^2\left(\sigma^2_{D_{c_i}}\frac{N_{c_i}}{N_{c_i}-1}H_{c_i} + \bar{X}^2_{c_i}(1-W_i)W_i\right)}$ $H_{c_i} = \sum_{k=a_i}^{b_i}\left(\frac{1}{k} - \frac{1}{N_{c_i}}\right)\frac{\binom{N_{c_i}}{k}\binom{N_i-N_{c_i}}{n_i-k}}{\binom{N_i}{n_i}}$ $W_i = \begin{cases} \sum_{k=a_i}^{b_i}\frac{\binom{N_{c_i}}{k}\binom{N_i-N_{c_i}}{n_i-k}}{\binom{N_i}{n_i}} & n_i \leqslant N_i - N_{c_i} \\ 1 & n_i > N_i - N_{c_i} \end{cases}$ $a_i = max\{1, n_i - N_i + N_{c_i}\}, b_i = min\{n_i, N_{c_i}\}$ (no computable unbiased estimator for $\sigma^2_{\theta_c}$ unless $F_c = 1$)	$\phi_c = \sum_{i=1}^{L}\frac{N_i}{n_i}\sum_{Y_j\in S_i} Y_j I_c(Y_j)$ (unbiased) $\sigma_{\phi_c}{=}\sqrt{\sum_{i=1}^{L}\frac{(N_i-n_i)}{n_i(N_i-1)}N_i^2\sigma^2_{Zi}}$ $\sigma^2_{\phi_c} = \mathbb{E}\left[\sum_{i=1}^{L}\frac{N_i(N_i-n_i)}{n_i-1}\sigma^2_{ZY_i}\right]$	$\gamma_c = \sum_{i=1}^{L}\frac{N_i}{n_i}\sum_{Y_j\in S_i} I_c(Y_j)$ (unbiased) $\sigma_{\gamma_c} = \sqrt{\sum_{i=1}^{L}\frac{(N_i-n_i)}{n_i(N_i-1)}N_i^2 F_{c_i}(1-F_{c_i})}$ $\sigma^2_{\gamma_c} = \mathbb{E}\left[\sum_{i=1}^{L}\frac{N_i(N_i-n_i)}{(n_i-1)}f_{c_i}(1-f_{c_i})\right]$	$\sigma^2_{D_{c_i}} = \mathbb{E}\left[\frac{1}{M}\sigma^2_{S_{c_i}}\right]$ $M = \frac{N_{c_i}}{N_{c_i}-1}\sum_{k=a_i}^{b_i}\frac{k-1}{k}\frac{\binom{N_{c_i}}{k}\binom{N_i-N_{c_i}}{n_i-k}}{\binom{N_i}{n_i}}$ $F_{c_i} = \mathbb{E}\left[f_{c_i}\right]$ $N_{c_i} = \mathbb{E}\left[N_i f_{c_i}\right]$ $\bar{X}_{c_i} = \mathbb{E}\left[\frac{\bar{Y}_{c_i}}{\sum_{k=a_i}^{b_i}\frac{\binom{N_{c_i}}{k}\binom{N_i-N_{c_i}}{n_i-k}}{\binom{N_i}{n_i}}}\right]$ $a_i = max\{1, n_i - N_i + N_{c_i}\}, b_i = min\{n_i, N_{c_i}\}$

Table 2: The error estimation results for different aggregate functions under different sampling strategies.

city	**avg_income**
New York	13,010
Los Angeles	34,560
Detroit	12,800
Ann Arbor	5,000

city	**avg_income**
New York	12,600 ± 700
Los Angeles	33,200 ± 1,100
Detroit	13,100 ±250

```
SELECT city, avg(income) as avg_income
FROM IRS
GROUP BY city
HAVING avg(income) > 13,000
```

The true answer to this query would be the following: However, using the same sample as the previous query, the approximate answer returned for this query will be as follows:

The problem with this approximation is self-evident. Certain tuples are missing from the output altogether, i.e., the row pertaining to New York city would have been present in the output of the query if executed against the original data. This is called *subset error*. This error cannot be easily prevented with stratified samples. Here, for example, the reason for New York's row being missing from the output is that our current estimate of its average income is 12,600 which is below the 13,000 cut-off point set by the HAVING clause.

Likewise, certain tuples should not have been present in the output of this query (i.e., the row pertaining to Detroit) as its true aggregate value in reality is 12,800 which is lower than 13,000. However, we have a group for Detroit in our output simply because our current estimate of Detroit's average income happens to be 13,100. This is the opposite side of the subset error, and is called the *superset error*.

Confidence intervals and other range-based accuracy guarantees are only apt at expressing the error of point estimates, such as the error of an aggregate function's output. However, as evident from these simple examples, subset and superset errors are of a different nature. These types of errors originate from the relational operators (e.g., selection, grouping, and projection) and are almost unheard of in the statistics literature. However, both subset and superset errors can be easily defined using the possible worlds semantics widely used in probabilistic query processing literature [23]. (See [67] as an example of the close connection between probabilistic and approximate databases). In particular, the subset and superset errors could be managed by computing the *probability of existence* for each possible output group [67]. For example, the following output will be far more informative (and semantically more consistent) than the previous output:

This additional probability accompanying each output tuple can easily cause further confusion and lead to an overwhelming number of output tuples. However, the purpose of this section is to simply point out the need for AQP systems to compute and treat these probabilities in a principled manner, in order to ensure correct semantics. Later, in Section 7, we discuss several approaches to hide this added complexity from the end users.

6 An Important Error Often Overlooked: Bias

As mentioned in Section 4, it is important to provide *unbiased estimates* to ensure that in the long run all the approximate answers are centered around the true answer. First, in Section 6.1, we show that the plug-in

city	avg_income
New York	12,600 ± 700
Los Angeles	33,200 ± 1,100
Detroit	13,100 ±250
Ann Arbor	NaN ± ∞

city	avg_income
New York	13,010
Los Angeles	34,560

approach can produce biased estimates even for relatively simple queries. Then, in Sections 6.3 and 6.4, we propose the use of bootstrap for estimating and correcting the bias.

6.1 Bias in the Plug-in Approach

Here, we show how the plug-in approach can lead to statistical bias even for queries that are relatively simple. Consider T3 as an instance of the IRS relation (shown in Table 3). Assume that the user wants to execute Q2 against T3. The true answer to this query will therefore be $\theta = Q_2(T_3) = (100K{+}200K{+}250K)/3 = 183.3K$.

For the sake of this toy example, let us assume that the AQP system decides to use the plug-in approach on a sample of size 3 to answer this query, i.e., $|S| = 3$. To compute the expected value of the returned approximation, $\mathbb{E}_{|S|=k}[\hat{\theta}_S]$, we need to consider all possible samples of size 3. Since $\binom{4}{3} = 4$, we can enumerate all these possibilities for this toy example:

$$\mathbb{E}_{|S|=3}(\theta(S)) = 100K \cdot (5/3 + 2 + 7/4 + 9/4)/4 = 191.7K$$

Thus, the bias b of our estimator for Q_2 is non-zero:

$$b = \mathbb{E}_{|S|=3}(\theta(S)) - \theta = (191.7 - 183.3) * 1000 = 8.4K \neq 0 \tag{3}$$

Note that this query was quite simple: it only consisted of two AVG aggregates, with no HAVING or even WHERE clause. Yet, the plug-in approach led us to a biased answer.[10] Almost all existing AQP systems have assumed (at least, implicitly) that the execution of the original query on a sample (i.e., the plug-in approach) yields an unbiased estimate of the true answer. Consequently, to this date, all error quantification efforts in the AQP community have focused on estimating the variance of the approximate answers. Given that the sampling error is often asymptotically normal, the estimated variance can be used to provide a confidence interval centered around the approximate answer. This is based on the assumption that the approximate answer from the plug-in approach is an unbiased estimate. Even more recent studies using bootstrap for error quantification assume that an unbiased estimate (e.g., in form of a re-written query) is provided by the user [67]. In other words, these techniques have only been used for estimating the variance but not the bias of the estimators.

In the next section, we explain how (non-parametric) bootstrap can be used to estimate the bias of the approximate answer produced by the plug-in approach. Once estimated, the bias can be subtracted from the original estimate to produce an unbiased approximate answer. For the unfamiliar reader, we first provide a brief introduction to the bootstrap theory.

[10] As an example of an even simpler query (i.e., with no nested sub-query) that still leads to bias, consider the following query: `SELECT AVG(X) FROM D WHERE X` $>$ `3` that is to be run against a sample of size $n = 1$ chosen from a table $D = \{0, 5\}$. There are two possible samples $\{0\}$ and $\{5\}$, with equal probabilities. These samples yield 0 and 5, respectively, as an approximate answer. Thus, the expected value of our approximation will be $\frac{0+5}{2} = 2.5$, while the true answer is 5.

city	avg_income
Los Angeles	33,200 ± 1,100
Detroit	13,100 ± 250

city	avg_income	probability_of_existence
New York	12,600 ± 700	0.98
Los Angeles	33,200 ± 1,100	0.999999
Detroit	13,100 ± 250	0.90
Ann Arbor	4,010 ± 150	0.0000001

6.2 Background: Bootstrap

In this section, we provide a brief overview of the *nonparametric bootstrap* (or simply the *bootstrap*), which has many applications in AQP systems. Bootstrap [26] is a powerful statistical technique traditionally designed for estimating the uncertainty of sample estimators. Consider an estimator θ (i.e., a query) that is evaluated on a sample S. This estimated value, denoted as $\theta(S)$, is a point-estimate (i.e., a single value), and hence, reveals little information about how this value would have changed had we used a different sample S'. This information is critical as the answer could be quite different from one sample to the next. Thus, S should be treated as a random variable drawn from the original population (dataset D in our case). Consequently, statisticians are often interested in measuring *distributional information* about $\theta(S)$, such as variance, bias, etc. Ideally, one could measure such statistics by (i) drawing many new samples, say $S_1, \cdots, S_k$ for some large k, from the same population D from which the original S was drawn, such that $|S| = |S_i|$, (ii) computing $\theta(S_1), \cdots, \theta(S_k)$, and finally (iii) inducing a distribution for $\theta(S)$ based on the observed values of $\theta(S_i)$. We call this the true distribution of $\theta(S)$. Figure 3(a) illustrates this computation. Given this distribution, any distributional information (e.g., variance) can be computed.

Unfortunately, in practice, the original dataset D cannot be easily accessed (e.g., due to its large size), and therefore, direct computation of $\theta(S)$'s distribution using the procedure of Figure 3(a) is impossible. This is where bootstrap [26] becomes useful. The main idea of bootstrap is simple: treat S as a proxy for its original population D. In other words, instead of drawing S_i's directly from D, generate new samples $S_1^*, \cdots, S_k^*$ by resampling from S itself. Each S_i^* is called a (bootstrap) replicate or simply a bootstrap. Each S_i^* is generated by drawing $n = |S|$ independently and identically distributed (I.I.D.) samples *with replacement* from S, and thus, some elements of S might be repeated or missing in each S_i^*. Note that all bootstraps have the same cardinality as S, i.e. $|S_i^*| = |S|$ for all i. By computing θ on these bootstraps, namely $\theta(S_1^*), \cdots, \theta(S_k^*)$, we can create an empirical distribution of $\theta(S)$. This is the bootstrap computation, which is visualized in Figure 3(b).

The theory of bootstrap guarantees that for a large class of estimators θ and sufficiently large k, we can use the generated empirical distribution of $\theta(S)$ as a consistent approximation of its true distribution. The intuition is that, by resampling from S, we emulate the original population D from which S was drawn. Here, it is sufficient (but not necessary) that θ be relatively smooth (i.e., Hadamard differentiable [26]) which holds for a large class of queries.[11] In practice, $k = 100$ is usually sufficient for achieving reasonable accuracy (k can also be tuned automatically; see [26]).

Advantages — Employing bootstrap has several key advantages for AQP systems. First, as noted, bootstrap delivers consistent estimates for a large class of estimators. Second, the bootstrap computation uses a "plug-in" principle; that is, we simply need to re-evaluate our query θ on S_i^*'s instead of S. Thus, we do not need to modify or re-write θ. Finally, individual bootstrap computations $\theta(S_1^*), \cdots, \theta(S_k^*)$ are independent from each other, and hence can be executed in parallel. This embarrassingly parallel execution model enables scalability

[11] Bootstrap has even been used for estimating the uncertainty of complex functions, such as machine learning classifiers (see [50]).

citizen_name	city	age	income
John	Detroit	38	100K
Alice	Ann Arbor	35	200K
Bob	Los Angeles	25	200K
Mary	Los Angeles	42	300K

Table 3: T_3: another instance of the IRS relation.

```
SELECT AVG(city_income)
FROM (
        SELECT city, AVG(income) as city_income
        FROM IRS
        GROUP BY city
);
```

Figure 2: Q_2: a simple query, computing the average of the average income across all cities.

by taking full advantage of modern many-core and distributed systems.

Disadvantages — The downside of bootstrap is its computational overhead, as hundreds or thousands of bootstrap trials are typically needed to obtain reliable estimates. However, several optimizations have been proposed for speeding up the bootstrap process. Pol et al. [56] have proposed Tuple Augmentation (TA) and On-Demand Materialization (ODM) for efficient generation of bootstrap samples, while Laptev et al. [39] have exploited the computation overlap across different bootstrap trials. Agarwal et al. [10] have shown that leveraging Poissonized resampling (as opposed to sampling with replacement) can yield significant speed ups. An alternative approach, called *Analytical Bootstrap Method* (ABM) [66, 67], avoids the Monte Carlo simulations altogether; instead, ABM uses a probabilistic relational model for the bootstrap process to automatically estimate the error (for a large class of SQL queries) to bypass the actual Monte Carlo simulation. Unlike the Poissonized resampling, ABM is limited to SQL operators and cannot handle user-defined aggregates.

6.3 Bias Estimation and Correction Using Bootstrap

The bias b of the plug-in approach is defined as:

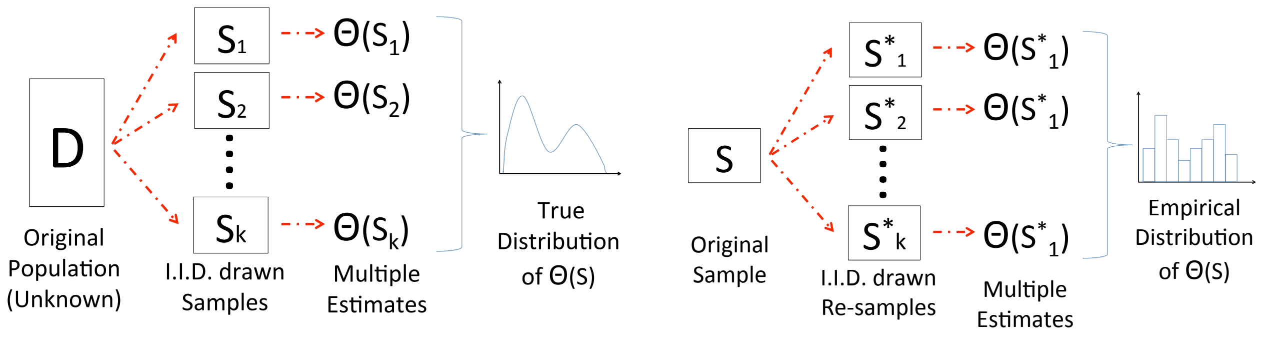
$$b = \mathbb{E}_{|S|=k}[\theta_S] - \theta = \mathbb{E}_{|S|=k}[q(S)] - q(D) \tag{4}$$

Using bootstrap theory, we can estimate $\mathbb{E}_{|S|=k}[q(S)] - q(D)$ by replacing S with S^* and D with S, as follows:

$$b \approx \mathbb{E}_{|S^*|=k}[q(S^*)] - q(S) \tag{5}$$

where S^* is a bootstrap resample of S and the expectation is taken over all bootstrap resamples. Since enumerating all possible bootstrap resamples is infeasible, we often restrict ourselves to m independent resamples of S for sufficiently large values of m. We denote these bootstrap resamples by $S_1, S_2, ..., S_B$, and take the average of $q(S_i)$ over $i \in \{1, 2, ..., m\}$:

$$b \approx \frac{1}{m}\sum_{i=1}^{m} q(S_i) - q(S) \tag{6}$$



(a) Ideal computation of $\theta(S)$'s distribution

(b) Bootstrap computation of $\theta(S)$'s distribution

Figure 3: Since the original data D may not be easily accessible to generate new samples S_i from, bootstrap uses resamples S_i^* of the original sample S instead, where $|S| = |S_i^*|$, to produce an empirical distribution for $\theta(S)$.

Choosing a large m (many bootstrap resamples) improves the accuracy of the bias estimation. The computational overhead of repeatedly evaluating q on each resample can be avoided by using the analytical bootstrap method (ABM) [67].

As a toy example, we apply this bias estimation technique to query Q2 and relation T3. Assume that $k = 3$ and that the chosen sample is $S = \{(200K, AnnArbor), (200K, LosAngeles), (300K, LosAngeles)\}$. Here, for brevity, we only display the income and city fields as the other fields do not affect the query results. For this small example, we can exhaustively enumerate all possible bootstrap resamples, their appearance probability, and their corresponding query result $q(S_i)$:

Table 4: Bootstrap resamples and their corresponding probability of appearance.

Resample	Probability of Appearance	$q(S_i)$
{(200K,Ann Arbor) , (200K,Ann Arbor) , (200K,Ann Arbor)}	1/27	200K
{(200K,Los Angeles), (200K,Los Angeles), (200K,Los Angeles)}	1/27	200K
{(300K,Los Angeles), (300K,Los Angeles), (300K,Los Angeles)}	1/27	300K
{(200K,Ann Arbor) , (200K,Los Angeles), (200K,Los Angeles)}	3/27	200K
{(200K,Ann Arbor) , (200K,Ann Arbor) , (200K,Los Angeles)}	3/27	200K
{(200K,Ann Arbor) , (200K,Ann Arbor) , (300K,Los Angeles)}	3/27	500K/2
{(200K,Ann Arbor) , (300K,Los Angeles), (300K,Los Angeles)}	3/27	500K/2
{(200K,Los Angeles), (200K,Los Angeles), (300K,Los Angeles)}	3/27	700K/3
{(200K,Los Angeles), (300K,Los Angeles), (300K,Los Angeles)}	3/27	800K/3
{(200K,Ann Arbor) , (200K,Los Angeles), (300K,Los Angeles)}	6/27	900K/4

Thus, we have:

$$\mathbb{E}_{|S|=3}[q(S^*)] = 100K(2*8/27 + 3*1/27 + 2.5*6/27 + 7/3*3/27 + 8/3*3/27 + 9/4*6/27) = 231.5K$$

Since $q(S) = (200K + 250K)/2 = 225K$, bootstrap will estimate the bias as $b \approx 231.5K - 225K = 6.5K$. This value is quite close to the true bias, which as computed in (3) is $b = 8.4K$. However, while this bias estimation strategy works well for the toy example at hand, it is important to note that this technique can be highly inaccurate when the sample size (i.e., $k = |S|$) or the number of resamples (i.e., m) are small. In the next section, we explain a reentering technique that can improve the accuracy of this bias estimation strategy.

6.4 Bias Estimation and Correction Using Bootstrap and Recentering

Let $S = \{X_1, X_2, ..., X_k\}$, where each X_j is a tuple. Also, let $S_1, S_2, ..., S_m$ be m bootstrap resamples of S. To apply the recentering method, we represent each S_i by a vector $V_i = (a_{i,1}, a_{i,2}, \cdots, a_{i,k})$, where $a_{i,j}$ is the number of times that tuple X_j is repeated in S_i. Thus, based on the definition of bootstrap resamples, $\sum_{j=1}^{k} a_{i,j} = k$. For example, if S_1 consists of $k-1$ repetitions of X_1 and one instance of X_k, then $V_1 = (k-1, 0, \cdots, 0, 1)$. We abuse our notation to use this vector representation: we replace $q(S_i)$ with $q(V_i)$. Let $\bar{V}$ denote the average of all V_i's:

$$\bar{V} = \frac{1}{m}\sum_{i=1}^{m} V_i$$

With this notation, the bootstrap with recentering technique [26] estimates the bias of an approximation as follows:

$$b \approx \frac{1}{m}\sum_{i=1}^{m} q(S_i) - q(\bar{V}) \tag{7}$$

The difference between equations (6) and (7) is that, in the latter, we have replaced $q(S)$ with $q(\bar{V})$.

The intuition behind why (7) is superior in accuracy to (6) is as follows. Note that $\frac{1}{m}\sum_{i=1}^{m} q(S_i)$ in (6) is used as an approximation of $\mathbb{E}_{|S|=k}[q(S^*)]$ in equation (5). However, unless we consider all (or many) possible resamples of S, this approximation can be quite inaccurate. Due to computational constraints, m may not be sufficiently large in practice. To compensate for this shortcoming, the recentering technique (conceptually) changes the distribution of the tuples in S to enforce that $\frac{1}{m}\sum_{i=1}^{m} q(S_i)$ is an accurate approximation of $\mathbb{E}_{|S|=k}[q(S^*)]$. In other words, before estimating the bias, we replace S with a new sample that fits the empirical distribution formed by our m bootstrap resamples. This new sample is our $\bar{V}$, which is formed by averaging the m bootstrapped resamples. It has been shown that this reentering strategy exhibits less variance than the technique described in Section 6.3 [26].

The toy example used in Section 6.3 enumerated all bootstrap resamples. When this is the case, it is easy to show that $q(S) = q(\bar{V})$, which causes the two techniques to produce identical estimates of the bias. However, in practice, enumerating all possible resamples is often infeasible, and thus, the reentering technique becomes superior. To demonstrate this, we reconsider the same query Q2 and sample S as used in the previous section, but limit ourselves to only $m = 6$ bootstrap resamples $S_1, S_2, \cdots, S_6$. These resamples, their V_i representation, and their corresponding $q(S_i)$ are all listed in the following table:

Table 5: Bootstrap resamples and their vector representations

S_i	V_i	$q(S_i)$
S_1 ={(200K,Ann Arbor) ,(300K,Los Angeles),(300K,Los Angeles)}	$V_1 = (1,0,2)$	$q(S^1) = 500K/2$
S_2 ={(300K,Los Angeles),(300K,Los Angeles),(300K,Los Angeles)}	$V_2 = (0,0,3)$	$q(S^2) = 300K$
S_3 ={(200K,Los Angeles),(300K,Los Angeles),(300K,Los Angeles)}	$V_3 = (0,1,2)$	$q(S^3) = 800K/3$
S_4 ={(200K,Los Angeles),(300K,Los Angeles),(300K,Los Angeles)}	$V_4 = (0,1,2)$	$q(S^4) = 800K/3$
S_5 ={(200K,Ann Arbor) ,(200K,Los Angeles),(300K,Los Angeles)}	$V_5 = (1,1,1)$	$q(S^5) = 900K/4$
S_6 ={(200K,Ann Arbor) ,(200K,Ann Arbor) ,(200K,Ann Arbor) }	$V_6 = (3,0,0)$	$q(S^6) = 200K$

Thus, using equation (6), the bias will be estimated as:

$$b \approx \frac{1}{6}\sum_{i=1}^{6} q(S_i) - q(S) = 251.4K - 225K = 26.4K$$

To use equation (7), we have $\bar{V} = (\frac{500K}{6}, \frac{100K}{2}, \frac{500K}{3})$, and:

$$q(\bar{V}) = 100K * \frac{2 + \frac{(1/2)*2+(5/3)*3}{(1/2)+(5/3)}}{2} = 238.5K$$

Now, we can use equation (7) to estimate the bias as:

$$b \approx \frac{1}{6}\sum_{i=1}^{6} q(S_i) - q(\bar{V}) = 251.4K - 238.5K = 12.9K$$

This estimate is closer to the real bias, which as computed in (3) is $b = 8.4K$.

In this example, although there is equal probability to bootstrap each of the tuples (200K, Ann Arbor), (200K, Los Angeles), and (300K, Los Angeles) from S, our bootstrap resamples included (300K, Los Angeles) more frequently than the other two tuples. Therefore, by recentering S to fit the empirical distribution formed by these 6 bootstrap resamples, the probability of bootstrapping (300K, Los Angeles) is enforced to be significantly higher than the probability of bootstrapping the other two items.

7 Communicating Error to Non-Statisticians

In general, for a SQL query with m aggregate columns in its SELECT clause, each output row is associated with $m + 1$ error terms: one error will capture the row's probability of existence, while others will capture the approximation quality of the aggregate columns in the row. Using confidence intervals as a common means of expressing aggregation error, we will need to (at least conceptually) append $2m + 1$ columns to the output relation of an approximate query, where each aggregation error will be captured using two columns: one for the confidence level α (e.g., 95%) and one for half of the interval width β. In other words, each output tuple T will have the following attributes:

$$T : B_1, \cdots, B_k, A_1, \cdots, A_m, \alpha_1, \cdots, \alpha_m, \beta_1, \cdots, \beta_m, p_e$$

where $B_1, \cdots, B_k$ are the non-aggregate columns in the output, $A_1, \cdots, A_m$ are the aggregate columns, α_i and β_i are the confidence level and half interval width for A_i, respectively. Finally, p_e is the probability of existence for tuple T (see Section 5). For example, assuming A_i is an unbiased estimate, with a probability of α, the true answer will be in the $[A_i - \beta_i, A_i + \beta_i]$ range.

While providing a rich set of accuracy measures, this scheme suffers from two major drawbacks:

1. Adding additional columns to the query output will break existing BI (Business Intelligence) tools designed to work with traditional database systems. Unless such tools are modified to expect (and ignore or use) the additional columns, they cannot work with an AQP system that modifies the output schema of the queries. In contrast, an AQP system that keeps the output schema intact will enjoy a greater degree of adoption: any off-the-shelf reporting and BI tool can be sped up by simply replacing its underlying RDBMS with such an AQP system, without the tool having to be aware of the fact that it is using an AQP database.

2. While a statistician might be able to understand, combine, and use these rich set of error statistics to perform sophisticated forms of inference, a typical database user will simply find a large number of errors associated with each row overwhelming. In fact, the majority of non-statisticians (e.g., developers and practitioners) may even find it difficult to interpret statistical error guarantees, such as confidence intervals and asymptotic errors.[12]

[12] One solution is to substitute β_i with $\frac{\beta_i}{|A_i|} \times 100\%$ (when $A_i > 0$) for a relative error or with $\pm\beta_i$ for an absolute error. Even better, one might use a graphical interface where these numbers are replaced with error bars [32]. However, grasping the statistical significance of all these error bars will still remain overwhelming.

On the other hand, there is clear advantage to provide and use these error estimates when relying on approximate answers for decision making. While there is a clear need for more research in this area (i.e., how to best communicate statistical errors to a non-statisticians), here we outline a basic solution to this dilemma, which we call a *high-level accuracy contract (HAC)*.

High-level Accuracy Contract (HAC) — First, the AQP system calculates all these errors but, by default, does not add them to the query output. The query, however, can access these error estimates by explicitly invoking a designated function. For example,

```
SELECT call_center_id, AVG(review) AS satisfaction
FROM customer_service_logs
WHERE satisfaction < 2.0
      AND existence_probability() > 0.95
      AND relative_error(satisfaction, 0.95, 0.3)
GROUP BY call_center_id
ORDER BY satisfaction
);
```

where `existence_probability()` and `relative_error()` are special functions provided by the AQP engine. This will simply allow users (and BI tools) to decide if and when they would like to use the error estimates. However, to allow practitioners and existing BI tools that are not ready or willing to explicitly invoke such functions, the AQP system provides a high-level accuracy contract, expressed as a single number ϕ, where $0\phi \leq 1$. Once a high-level accuracy contract is set to a particular ϕ, the AQP system guarantees that any results returned to the users will be at least $\phi \times 100\%$ accurate. This is enforced as follows.

The AQP system simply omits every output tuple whose probability of existence is below ϕ. However, to deal with an aggregate value that does not meet this requested HAC[13] there are several policies that the AQP system can adopt:

- *Do nothing.* The AQP system simply returns all the aggregate values (possibility with a different display color if a graphical terminal is used).
- *Use a special symbol.* The system replaces the aggregate value with a special value, e.g., `NULL`, -1, or other user-specified values.
- *Drop the row.* The system drops the entire row if any of the aggregate columns in that row do not meet the required HAC.
- *Fail.* The system drops the entire output relation and throws an exception (e.g., through a SQL error) if any of the aggregate columns in any of the rows do not meet the required HAC.

The user can control the AQP system's behavior by choosing any of these policies, with 'do nothing' being the most lenient approach and 'fail' being the strictest behavior. In the latter, the user can decide whether to re-run the query with a more lenient policy or with a larger sample size, or simply resort to exact query evaluation. The downside of the 'drop the row' policy is that it will affect the internal logic of the BI tools if they keep track of the row count of the output relation. The 'use a special symbol' does not change the output cardinality, however, it will need a special symbol that is not present in the original data. The main advantages of the HAC approach is that (i) it allows AQP systems to be used as a drop-in solution for existing BI tools, (ii) enables practitioners and developers to express their required level of accuracy in an intuitive fashion, i.e., as a single percentage, (iii) provides a range of intuitive policies to cater to different levels of accuracy concerns, and (iv) allows advanced users to still access and use the detailed error statistics if they want to.

[13] An aggregate value does not meet the HAC if $\frac{\beta_i}{max\{|A_i|,\epsilon\}} < \phi$, where $\epsilon > 0$ is a small constant to prevent unduly small values of A_i from causing large relative errors. α_i can be also set to ϕ or to a default value, e.g., 0.95 or 0.99.

8 Conclusion

With the Big Data on the rise, there has been much demands in the recent years for building and using Approximate Query Processing (AQP) systems. Despite the rich history of database research in this area, there are many technical subtitles and open challenges that need to be properly addressed in order to build sound, efficient, and deployable AQP systems. In this chapter, for the first time, we focused on these subtle challenges, ranging from the marketing opportunities for AQP systems to their general anatomy and architecture. Considering that most database developers and practitioners are not statisticians, we provided a self-contained and self-explanatory cheatsheet for error estimation under different sampling strategies. At the same time, we explained a few major types of error in these systems—that tend to be easily overlooked—using simple examples. Finally, we introduced our *high-level accuracy contract (HAC)* approach to prevent the complexity of different types of error from overwhelming the end user of an AQP system.

Acknowledgements

The authors are grateful to Christopher Jermaine for encouraging them to write this chapter, to Jia Deng and Jacob Abernethy for their great feedback, to Suchee Shah for her helpful comments, and to many researchers in the database community who have helped advance the field of AQP systems through their contributions over the years. This work was in part supported by Amazon AWS, Microsoft Azure, and NSF (Grant No. CPS 1544844).

References

[1] Databricks. `http://databricks.com/`.

[2] Good data. `http://www.gooddata.com/`.

[3] IDC report. `http://tinyurl.com/lvup9e4`.

[4] Presto: Distributed SQL query engine for big data. `https://prestodb.io/docs/current/release/release-0.61.html`.

[5] SnappyData. `http://www.snappydata.io/`.

[6] Tableau software. `http://www.tableausoftware.com/`.

[7] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.

[8] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *ACM SIGMOD*, 2000.

[9] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[10] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[11] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[12] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 2012.

[13] M. Al-Kateb and B. S. Lee. Stratified Reservoir Sampling over Heterogeneous Data Streams. In *SSDBM*, 2010.

[14] S. Arya and D. M. Mount. Approximate range searching. In *SCG*, 1995.

[15] B. Babcock and S. Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In *SIGMOD*, 2005.

[16] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *VLDB*, 2003.

[17] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[18] V. Chandrasekaran and M. I. Jordan. Computational and statistical tradeoffs via convex relaxation. *CoRR*, abs/1211.1073, 2012.

[19] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, 2000.

[20] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[21] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4, 2012.

[22] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55, 2005.

[23] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4), 2007.

[24] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. Sampling algorithms and coresets for \ell_p regression. *SIAM Journal on Computing*, 38, 2009.

[25] A. Dobra, C. Jermaine, F. Rusu, and F. Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2(1), 2009.

[26] B. Efron and R. Tibshirani. *An introduction to the bootstrap*, volume 57. CRC press, 1993.

[27] D. Fisher. Incremental, approximate database queries and uncertainty for exploratory visualization. In *LDAV*, 2011.

[28] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS*, 2008.

[29] R. Gemulla. Sampling algorithms for evolving datasets. 2008.

[30] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *KDD*, 2005.

[31] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, 2004.

[32] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[33] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47, 1952.

[34] Y. Hu, S. Sundara, and J. Srinivasan. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, 2009.

[35] S. Joshi and C. Jermaine. Robust Stratified Sampling Plans for Low Selectivity Queries. In *ICDE*, 2008.

[36] S. Joshi and C. Jermaine. Sampling-Based Estimators for Subset-Based Queries. *VLDB J.*, 18(1), 2009.

[37] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 2015.

[38] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, 1995.

[39] N. Laptev, K. Zeng, and C. Zaniolo. Early Accurate Results for Advanced Analytics on MapReduce. *PVLDB*, 5(10):1028–1039, 2012.

[40] J. Laurikkala. *Improving identification of difficult small classes by balancing class distribution*. Springer, 2001.

[41] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, 2013.

[42] S. Matusevych, A. Smola, and A. Ahmed. Hokusai-sketching streams in real time. *arXiv preprint arXiv:1210.4891*, 2012.

[43] X. Meng. Scalable simple random sampling and stratified sampling. In *ICML*, 2013.

[44] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM, 1968.

[45] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[46] B. Mozafari. Verdict: A system for stochastic query planning. In *CIDR, Biennial Conference on Innovative Data Systems*, 2015.

[47] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. CliffGuard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[48] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. Cliffguard: An extended report. Technical report, University of Michigan, Ann Arbor, 2015.

[49] B. Mozafari and N. Niu. A handbook for building an approximate query engine. Technical report, University of Michigan, Ann Arbor, 2015.

[50] B. Mozafari, P. Sarkar, M. J. Franklin, M. I. Jordan, and S. Madden. Scaling up crowd-sourcing to very large datasets: A case for active learning. *PVLDB*, 8, 2014.

[51] B. Mozafari and C. Zaniolo. Optimal load shedding with aggregates and mining queries. In *ICDE*, 2010.

[52] C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *SIGMOD*, 2009.

[53] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.

[54] Y. Park, M. Cafarella, and B. Mozafari. Neighbor-sensitive hashing. *PVLDB*, 2015.

[55] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. *CoRR*, 2015.

[56] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *In SIGMOD Conference Proceedings*, 2005.

[57] N. Potti and J. M. Patel. DAQ: a new paradigm for approximate query processing. *PVLDB*, 8, 2015.

[58] C. Qin and F. Rusu. Pf-ola: a high-performance framework for parallel online aggregation. *Distributed and Parallel Databases*, 2013.

[59] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*, 2011.

[60] H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, and C. Zaniolo. SMM: A data stream management system for knowledge discovery. In *ICDE*, 2011.

[61] S. K. Thompson. *Sampling*. Wiley series in probability and statistics. J. Wiley, 2012.

[62] Y. Tillé. *Sampling algorithms*. Springer, 2011.

[63] S. Vrbsky, K. Smith, and J. Liu. An object-oriented semantic data model to support approximate query processing. In *Proceedings of IFIP TC2 Working Conference on Object-Oriented Database Semantics*, 1990.

[64] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.

[65] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.

[66] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. Abs: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.

[67] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.