PilotDB: Database-Agnostic Online Approximate Query Processing with A Priori Error Guarantees

YUXUAN ZHU, University of Illinois Urbana Champaign, USA
TENGJUN JIN, University of Illinois Urbana Champaign, USA
STEFANOS BAZIOTIS, University of Illinois Urbana Champaign, USA
CHENGSONG ZHANG, University of Illinois Urbana Champaign, USA
CHARITH MENDIS, University of Illinois Urbana Champaign, USA
DANIEL KANG, University of Illinois Urbana Champaign, USA

After decades of research in approximate query processing (AQP), its adoption in the industry remains limited. Existing methods struggle to simultaneously provide user-specified error guarantees, eliminate maintenance overheads, and avoid modifications to database management systems. To address these challenges, we introduce two novel techniques, TAQA and BSAP. TAQA is a two-stage online AQP algorithm that achieves all three properties for arbitrary queries. However, it can be slower than exact queries if we use standard row-level sampling. BSAP resolves this by enabling block-level sampling with statistical guarantees in TAQA. We implement TAQA and BSAP in a prototype middleware system, PilotDB, that is compatible with all DBMSs supporting efficient block-level sampling. We evaluate PilotDB on PostgreSQL, SQL Server, and DuckDB over real-world benchmarks, demonstrating up to 126× speedups when running with a 5% guaranteed error.

CCS Concepts: • **Information systems** → **Online analytical processing**; *Middleware for databases*; *Data analytics*; • **Mathematics of computing** → *Statistical paradigms.*

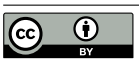
Additional Key Words and Phrases: approximate query processing, sampling

ACM Reference Format:
Yuxuan Zhu, Tengjun Jin, Stefanos Baziotis, Chengsong Zhang, Charith Mendis, and Daniel Kang. 2025. PilotDB: Database-Agnostic Online Approximate Query Processing with A Priori Error Guarantees. In *Proceedings of ACM Management of Data (SIGMOD '25).* ACM, New York, NY, USA, Article 198, 28 pages. https://doi.org/10.1145/3725335

1 Introduction

Approximate query processing (AQP) is widely studied to accelerate queries in big data analytics [1, 2, 5, 7, 11, 12, 16, 27, 32, 33, 57, 65, 71, 74, 77, 79, 92, 104, 106]. Although AQP has been extensively explored in academia, its adoption is still limited in practice [21, 62, 94]. Prior research demonstrates three properties that are crucial for real-world AQP applications: (**P1**) guaranteeing user-specified errors before the query is executed (i.e., a priori error guarantees) [5, 13, 27, 57, 65, 74, 104], (**P2**)

Authors' Contact Information: Yuxuan Zhu, University of Illinois Urbana Champaign, Champaign-Urbana, USA, yxx404@illinois.edu; Tengjun Jin, University of Illinois Urbana Champaign, Champaign-Urbana, USA, tengjun2@illinois.edu; Stefanos Baziotis, University of Illinois Urbana Champaign, Champaign-Urbana, USA, sb54@illinois.edu; Chengsong Zhang, University of Illinois Urbana Champaign, Champaign-Urbana, USA, cz81@illinois.edu; Charith Mendis, University of Illinois Urbana Champaign, Champaign-Urbana, USA, charithm@illinois.edu; Daniel Kang, University of Illinois Urbana Champaign, Champaign-Urbana, USA, ddkang@illinois.edu.



SIGMOD '25, Berlin, Germany

ACM ISBN 2836-6573/2025/6
https://doi.org/10.1145/3725335

Table 1. Characteristics of state-of-the-art AQP systems and algorithms. Online AQP inherently eliminates sample maintenance overhead. PilotDB is the first one that achieves a priori error guarantees (P1), eliminates maintenance overheads (P2), and avoids DBMS modifications (P3), at the same time.

AQP System	A Priori Error Guarantees (P1)	w/o Maintenance Overhead (P2)	w/o Modifying DBMSs (P3)
BlinkDB [5]	✓	×	×
Taster [74]	✓	×	×
Sample+Seek [27]	✓	×	×
Quickr [57]	✓	✓	×
BAQ [65]	✓	×	✓
VerdictDB [79]	×	×	✓
DBest [66]	×	✓	✓
PilotDB	✓	✓	✓

no maintenance overheads [57, 66], and (**P3**) not modifying the underlying database management system (DBMS) [65, 66, 68, 79].

However, none of the existing systems or algorithms achieves all three properties simultaneously (Table 1). We can further categorize these techniques into two types: *offline* methods that pre-compute samples and *online* methods that generate samples at query time.

Existing offline AQP methods require maintenance overheads [1, 2, 5, 7, 11, 12, 27, 33, 65, 79], sacrificing P2 and leading to limitations in deployment. Offline methods operate in two stages. In the offline stage, they pre-compute data samples based on expected workloads [5, 65]. At query time, offline samples that satisfy the error specification are selected to answer the query. Consequently, when data or workloads are updated, re-computations and/or manual inspections are required to maintain a priori error guarantees [5, 27, 65]. The cumulative costs of this maintenance can be a significant overhead that discourages deployment and commercial adoption [13, 70].

Although online methods eliminate maintenance overheads (P2) [9, 36, 39, 53, 57, 93, 105], existing online AQP algorithms require modifying DBMSs to achieve a priori error guarantees [57], sacrificing P3. These algorithms depend on sophisticated samplers and optimization logic for query acceleration and error guarantees [57]. However, these techniques are tightly integrated with DBMSs and lack widespread support. AS a result, adopting them requires modifying existing DBMSs, which can be infeasible for commercial applications [68, 69, 79].

In this paper, we propose two novel techniques to simultaneously achieve P1, P2, and P3, while accelerating queries compared to executing exact queries. First, we introduce a two-stage online AQP algorithm, TAQA, that achieves all three properties through query rewriting. However, TAQA alone cannot accelerate query processing due to sampling overhead. Therefore, we develop BSAP, a set of statistical techniques that formalizes block-level sampling—a more efficient sampling method than the widely used row-level sampling—with error guarantees. Finally, we build a middleware AQP system, PilotDB, which implements TAQA and BSAP.

TAQA. Our online AQP algorithm, TAQA, achieves all three properties through two stages of query rewriting and online sampling. In the first stage, we rewrite the input query and execute it to determine the optimal sampling plan that (1) satisfies the user's error specification and (2) minimizes the execution cost. In the second stage, we rewrite the input query with the optimal sampling plan and execute it, delivering results directly to users. For both stages, the rewritten queries leverage existing samplers in the DBMS.

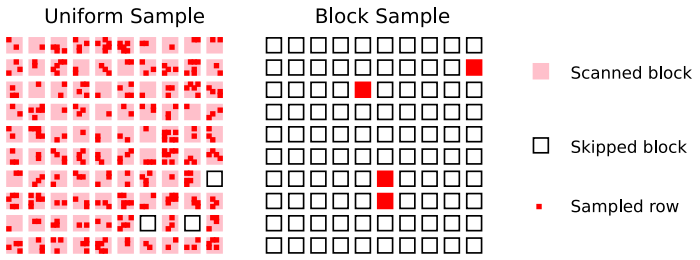


Fig. 1. Sampling 3% data from a table with a block size of 100 rows: in expectation, the row-level method requires scanning approximately 95% blocks, while the block-level method scans approximately 3% blocks.

However, naively using samplers of existing work in TAQA either fails to accelerate queries or requires modifying DBMSs. Specifically, row-level samplers are inefficient in databases that read data at the block level, resulting in query latencies as high as exact queries (§4.1) [6, 89]. This is especially the case for analytical queries where data scanning is often the primary latency bottleneck [15, 98]. To address the inefficiency of row-level samplers, researchers have developed more efficient sampling techniques, such as index-assisted sampling [99, 109]. Unfortunately, these techniques require modifying DBMSs and are not widely supported, sacrificing P3.

As a promising solution to accelerate TAQA without modifying DBMSs, block sampling, which samples data at the block level, achieves high efficiency by skipping non-sampled blocks (Figure 1).[1] Quantitatively, sampling 0.01% data from a table with 6B rows using block sampling can be up to 500× faster than uniform row-level sampling (Figure 5). Furthermore, our analysis reveals that for the same error specification, the sample size required for uniform block sampling can be comparable or even smaller than that of uniform row-level sampling (§4.1).[2]

BSAP. Although block sampling has been included in the ISO standard SQL [61] and is widely supported [22, 24, 29, 42, 50, 87, 100], existing error analysis techniques are insufficient to handle block sampling in nested or Join queries. Naively applying existing techniques can lead to errors up to 52× higher than the user-requested error (§5.2), sacrificing P1. We introduce new statistical techniques, BSAP, to formalize block sampling in approximate queries with statistical error guarantees. For deep nested queries, we establish sampling equivalence rules to reason about the interaction between block sampling and relational operations. For Join queries, we analyze the asymptotic distribution and the variance of the Join result over block samples, extending the standard central limit theorem (CLT) that fails when block sampling is executed on multiple tables [14, 47, 108].

With BSAP, we can further accelerate prior online AQP systems. In particular, we can use block sampling to replace the heavily-used uniform row-level sampling [57], while preserving the error guarantees. We empirically show that BSAP can accelerate QUICKR by up to 60× (§5.4) and TAQA by up to 219× (§5.5), compared to uniform row-level sampling.

We build a prototype middleware AQP system, PILOTDB, that implements TAQA and BSAP. We evaluate PILOTDB on various DBMSs, showing that it can achieve substantial query speedups on diverse benchmarks, including TPC-H [23], Star Schema Benchmark [75], ClickBench [20], Instacart [52, 79], and DSB [26]. When connected to transactional databases—PostgreSQL [96] and SQL Server [67]—PILOTDB achieves up to 126× speedup. When connected to an analytical

[1]Throughout the paper, "block" refers to the minimum unit of data accessing in the storage layer.
[2]In an extreme case where the variance and expectation of a block is similar to the entire dataset, sampling one block can be sufficient for a small error rate.

database—DuckDB [88]—PilotDB achieves up to 13× speedup. Furthermore, PilotDB consistently achieves a priori error guarantees across various settings.

We summarize our contributions as follows:

(1) We propose TAQA, an online AQP algorithm that simultaneously achieves P1, P2, and P3 (§3).
(2) We develop BSAP, a set of statistical techniques that enable block sampling to answer approximate nested and Join queries with statistical guarantees (§4).
(3) We build and evaluate PilotDB, which implements TAQA and BSAP, achieving a priori error guarantees and up to two orders of magnitude speedup on various DBMSs (§5).

2 Overview

In this section, we present an overview of PilotDB. We first discuss the background and challenges of building PilotDB (§2.1). Next, we introduce the workflow of PilotDB (§2.2). Finally, we describe the types of queries (§2.3) and the semantics of errors (§2.4) that PilotDB supports.

2.1 Background and Challenges

In Table 1, we summarize the characteristics of state-of-the-art AQP systems in terms of a priori error guarantees (P1), maintenance overheads (P2), and DBMS modifications (P3). We then present the background and challenges of simultaneously achieving P1, P2, and P3 from the perspective of algorithmic and statistical techniques.

Algorithmic Challenges. Given a query and an error specification (§2.4), an AQP algorithm must plan sampling properly to achieve a priori error guarantees (P1). A sampling plan specifies the sampling method, table(s) to sample, and the sample size, which determines whether the query can be accelerated and whether the error specification can be satisfied. To determine the sampling plan, prior work either pre-computes samples based on knowledge of the query workload [5, 27, 65, 74] or inserts samplers to the query plan at query time based on online statistics [57]. However, these methods break either P2 or P3. The pre-computation method requires maintenance efforts to refresh samples when data changes [5, 27, 65, 74], while the method of inserting samplers at query time requires modifying the execution and optimization logic of DBMSs [57].

We aim to resolve the tension among P1, P2, and P3. As we explained, the key challenge is to determine the sampling plan without pre-computation or controlling the query execution. To address it, we propose a novel online AQP algorithm that processes a query in two stages to plan and execute sampling (§3).

Statistical Challenges. Confidence intervals derived from statistical theories, such as CLT, are widely used to analyze errors of AQP [3–5, 12, 27, 28, 36–39, 46, 53, 57, 65, 72–74, 79, 103]. However, deriving valid confidence intervals for AQP with block sampling brings up two challenges that are not addressed in existing literature.

First, we need to analyze errors when there are intermediate relational operations (e.g., Join and Group By) between block sampling and aggregations. Prior work studies confidence intervals for simple Select-Aggregation query with block sampling [44, 78]. However, realistic queries often have more relational operations after executing the sampling, which can potentially affect the confidence interval computation. Previous research on interactions between row-level sampling and relational operations cannot be applied to block-level sampling because it cannot handle the dependence of rows from the same block [57, 72]. In this work, we propose sampling equivalence rules that establish the commutativity between block sampling and relational operations (§4.2), allowing us to analyze errors of deep approximate queries that use block sampling.

```
SELECT l_returnflag, l_linestatus,
       SUM(l_extendedprice) as agg_1, AVG(l_extendedprice) as agg_2
FROM lineitem
WHERE l_shipdate <= date '1998-12-01' - interval '90 day'
GROUP BY l_returnflag, l_linestatus
-- error specification
ERROR WITHIN 5%
PROBABILITY 95%
```

Fig. 2. An example query supported by PilotDB.

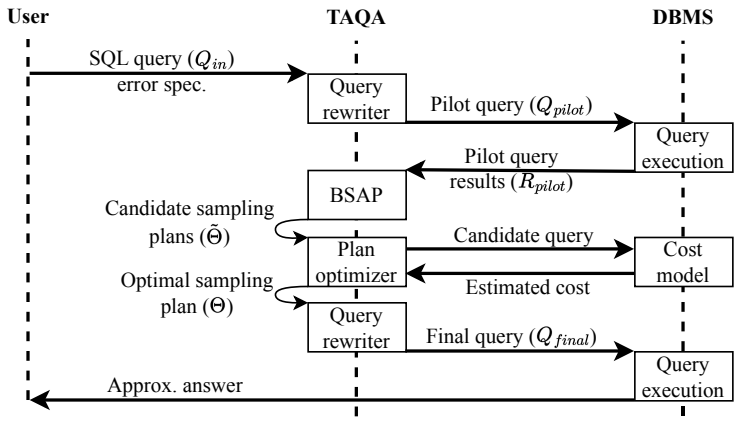


Fig. 3. Workflow of PilotDB.

Second, we need to obtain valid confidence intervals when multiple tables in a Join query are sampled at the block level. Existing literature studies the asymptotic distribution of the Join result when tables are sampled with the same sample size [38]. However, it is insufficient in our case because we target a richer sampling space where sample sizes for tables can be arbitrarily different. To address that, we extend the theoretical result of Hass et al. [38] to a general form and derive an estimation of the upper bound of the sampling variance to achieve error guarantees (§4.3).

Those challenges are crucial to formalizing block sampling in AQP with error guarantees. We unify our theoretical results into BSAP, which can also be used to further accelerate other online AQP algorithms beyond PilotDB (§5.4).

2.2 Workflow

PilotDB operates as a middleware between the user and the DBMS. As shown in Figure 2, users may issue queries to PilotDB in the same way that they interact with a DBMS except that PilotDB takes additionally the error specification (§2.4) as input and produces an approximate answer.

On receiving the user's input, PilotDB processes it with the TAQA algorithm. TAQA first rewrites the input query Q_{in} into a pilot query Q_{pilot} that computes necessary statistics for error analysis. Then, TAQA issues Q_{pilot} to the DBMS and obtains the pilot result R_{pilot}. Based on R_{pilot} and the error specification, TAQA incorporates BSAP to decide whether Q_{in} can be approximated efficiently using block sampling. Specifically, TAQA uses BSAP to analyze the error (§4) and generates a set of candidate sampling plans, $\tilde{\Theta}$, that guarantee the error specification (§3.1). If TAQA cannot identify

any feasible sampling plans, PilotDB will proceed to execute the original query Q_{in}. Otherwise, TAQA interacts with the cost model of the DBMS to determine the optimal sampling plan Θ that minimizes the estimated cost (§3.2). Finally, TAQA rewrites Q_{in} to a final query Q_{final} based on Θ and issues Q_{final} to the DBMS. We visualize this workflow in Figure 3.

2.3 Supported Queries

PilotDB is designed to answer all queries the underlying DBMS supports by directly executing the original query on the DBMS when necessary. There are two cases where PilotDB may fail to accelerate a query: (1) TAQA does not support the query or (2) the cost model indicates that block sampling cannot accelerate the query. In the first case, PilotDB directly passes the query to the DBMS without intercepting it. In the second case, PilotDB intercepts the query processing with TAQA but executes the original query eventually.

PilotDB tries to intercept and accelerate arbitrary aggregation queries using TAQA with the following exceptions. First, PilotDB does not support non-linear aggregates, (e.g., `COUNT DISTINCT`, `MAX`, and `MIN`), or aggregates in Group By clauses (e.g., `GROUP BY COUNT(*)`). These queries are challenging for AQP and not supported in prior techniques [5, 27, 57, 79]. Moreover, if any subqueries are correlated [91], PilotDB tries to replace correlated subqueries with Joins using pre-defined rules. If PilotDB fails to apply rules, it falls back to executing the exact query, since executing the pilot query is expensive if the query is correlated [79].

PilotDB may fail to accelerate extremely selective queries or queries with a large group cardinality. These two cases are challenging to support for sampling-based AQP [57, 79]. However, prior online AQP may still use sampling on those queries, which results in errors larger than the user-specified error [57]. By contrast, PilotDB incorporates sampling plan optimization (§3.2) to determine that sampling is infeasible or not efficient for such queries. PilotDB defaults to executing exact queries in this case.

2.4 Error Specifications and Semantics

Finally, we describe how users can specify error requirements in PilotDB and then define the statistical semantics of the error specification. PilotDB allows users to specify a maximum relative error for all aggregates together with a probability or confidence, which are the same specifications prior work allows [5, 57, 74, 104].

Taking the query in Figure 2 as an example, the error specification means that the probability of relative errors of `agg_1` and `agg_2` being simultaneously less than 5% is at least 95%. Generally, consider a query with k aggregations and m groups, resulting in a set of $k \cdot m$ aggregates: $\{\mu_{i,j} | 1 \leq i \leq k, 1 \leq j \leq m\}$. We denote $\hat{\mu}_{i,j}$ as the estimate of the aggregate $\mu_{i,j}$. An error specification with a relative error e and confidence p means that PilotDB will output a set of estimated aggregates such that the probability that all estimates simultaneously have a relative error no greater than e (i.e., the probability of the intersection of events) is at least p. Namely,

$$\mathbb{P}\left[\bigcap_{1\leq i\leq k, 1\leq j\leq m} \left|\frac{\mu_{i,j} - \hat{\mu}_{i,j}}{\mu_{i,j}}\right| \leq e\right] \geq p \tag{1}$$

Our error specification limits the joint probability of all estimates having unexpected errors across aggregations and groups. This is stronger and more intuitive for users to interpret than prior work that only reasons about the error of each estimate independently [5, 57]. We will tackle the joint probability in the next section.

3 Two-Stage Query Approximation

In this section, we focus on addressing the algorithmic challenges mentioned in Section 2.1. We introduce our two-stage query approximation algorithm to answer the following three questions:

(1) How can we find a *valid* sampling plan that satisfies the user's error specification (§3.1)?
(2) How can we find an *efficient* sampling plan that minimizes the execution cost of TAQA (§3.2)?
(3) How can we achieve (1) and (2) via query rewriting (§3.3)?

3.1 Sample Planning via Pilot Query Processing

We determine sampling plans that satisfy the user's error specification by executing a pilot query that inspects the statistical property of the input query. To understand what should be inspected through the pilot query, we first parametrize the sampling plan.

Given a query with k tables, a sampling plan should specify the sampling method and corresponding sampling parameters for each table. To avoid modifying the DBMSs, we use Bernoulli sampling where each unit (e.g., a row or a block) is independently selected with a fixed sampling rate or probability θ without replacement. In many DBMSs [61], row-level Bernoulli sampling is supported through the `TABLESAMPLE BERNOULLI` clause while block-level Bernoulli sampling is expressed via `TABLESAMPLE SYSTEM`.

Although Bernoulli sampling produces variable sample sizes, we can still provide error guarantees by parameterizing the k-table sampling plan into a list of k sampling rates: $\Theta = [\theta_1, \ldots, \theta_k]$. This approach allows us to account for the variability in sample sizes when deriving guarantees. In the rest of this section, we present the statistical intuition and formulation underlying this approach.

Statistical Intuition. Consider the scenario where the query involves one aggregation computed on one group. We can calculate the confidence interval to analyze the relative error of the estimate. Suppose we have a population with mean μ that is estimated with a sample mean $\hat{\mu}$. We denote $Var[\hat{\mu}]$ as the variance of $\hat{\mu}$. We can establish the following CLT-based confidence interval for μ:

$$\mathbb{P}\left[\hat{\mu} - z_{(1+p)/2}\sqrt{Var[\hat{\mu}]} \leq \mu \leq \hat{\mu} + z_{(1+p)/2}\sqrt{Var[\hat{\mu}]}\right] \geq p \tag{2}$$

where $z_{(1+p)/2}$ is the $(1+p)/2$ percentile of the standard normal distribution. When μ is positive, Inequality 2 can be equivalently converted to an inequality on the relative error of $\hat{\mu}$:

$$\mathbb{P}\left[\left|\frac{\hat{\mu} - \mu}{\mu}\right| \leq \frac{z_{(1+p)/2}\sqrt{Var[\hat{\mu}]}}{\mu}\right] \geq p$$

That is, to satisfy the error specification with a maximum relative error e and a confidence p, it is sufficient to ensure that

$$z_{(1+p)/2} \cdot \sqrt{Var[\hat{\mu}]} \cdot \mu^{-1} \leq e \tag{3}$$

With Inequality 3, we observe that determining μ and $Var[\hat{\mu}]$ is the key to satisfying the error specification. However, μ and $Var[\hat{\mu}]$ are unknown unless we execute the input query. To address this, prior work maintains pre-computed samples [5, 27, 65] or modifies DBMSs to monitor statistics during the query execution [57]. In TAQA, we estimate μ and $Var[\hat{\mu}]$ by executing a pilot query that is dynamically rewritten from the input query.

To minimize the latency overhead, the pilot query samples the table that is most expensive to load. This is achieved in two steps. First, PilotDB obtains an execution plan of the original query to inspect the table loading method used by the DBMS. A table is considered as a candidate to sample

if the DBMS uses scanning.[3] Second, PilotDB queries the estimated table cardinality maintained by the DBMS and samples the largest table that will be scanned.

From the pilot query result, we can estimate the lower bound of μ and the upper bound of $Var[\hat{\mu}]$ where $\hat{\mu}$ will be computed using a sampling plan Θ in the final query. We first focus on sampling one table in the final query and then address sampling multiple tables in Section 4.3. Assuming $\hat{\mu}$ is sub-Gaussian,[4] these bounds are estimated using standard technique based on the CLT [43], a widely used approach in AQP [5, 27, 57, 79]. The sub-Gaussian assumption implies that $\hat{\mu}$ has a fast decaying tail bounded above by a Gaussian distribution. Then, the analytical expression of the bounding distribution can be derived using CLT asymptotically.

For instance, given sample size n and sample variance $\hat{\sigma}$, we have $\frac{\mu-\hat{\mu}}{\hat{\sigma}/n} \to t_{n-1}$ as $n \to \infty$. With sufficiently large n, we have:

$$\mathbb{P}\left[\mu \geq \hat{\mu} - t_{n-1,1-\delta} \cdot \hat{\sigma} \cdot n^{-1}\right] \geq 1 - \delta$$

where δ is a pre-specified failure probability and $t_{n-1,1-\delta}$ is $1-\delta$ percentile of Student's t distribution. We can obtain the upper bound of $Var[\hat{\mu}]$ similarly since the ratio between the variance σ^2 and its estimate $\hat{\sigma}^2$ converges to chi-squared distribution: $(n-1)\hat{\sigma}^2/\sigma^2 \to \chi^2_{n-1}$. Furthermore, as n follows the binomial distribution $Bin(N, \theta)$, we can estimate the lower bound of n given the upper bound of the population size N that is obtained using the pilot query result.

However, this is not sufficient to guarantee the confidence p since these bounds obtained from statistical distributions are *probabilistic*. A probabilistic bound can fail with a *controllable* probability [43]. Therefore, to ensure the overall validity, we adjust the confidence p based on the failure probability of all probabilistic bounds we used in the derivation, which leads to the confidence p' in Procedure 1.

Formal Description. We formalize the intuition as follows.

Procedure 1. *Consider an input query Q_{in} that computes a linear aggregate μ. Suppose a user specifies a maximum relative error e and a confidence p. In the first stage, we rewrite Q_{in} into a pilot query Q_{pilot} with sampling rate θ_p. Based on the result of Q_{pilot}, we can calculate (1) L_μ: a probabilistic lower bound of μ, and (2) $U_V[\Theta]$: a probabilistic upper bound of $Var[\hat{\mu}]$ given a sampling plan Θ. Namely, with pre-specified failure probabilities δ_1 and δ_2, we can obtain the following inequalities:*

$$\mathbb{P}\left[\mu \geq L_\mu\right] \geq 1 - \delta_1 \tag{4}$$

$$\mathbb{P}\left[Var[\hat{\mu}] \leq U_V[\Theta]\right] \geq 1 - \delta_2 \tag{5}$$

We find a sampling plan Θ such that the following inequality holds

$$z_{(1+p')/2} \cdot \sqrt{U_V[\Theta]} \cdot L_\mu^{-1} \leq e \tag{6}$$

where p' is the adjusted confidence based on the probabilities in Inequalities 4 and 5:

$$p' = p + \delta_1 + \delta_2$$

Procedure 1 involves three tunable parameters: θ_p, δ_1, and δ_2. Intuitively, a smaller θ_p reduces overhead of executing Q_{pilot}, while a larger θ_p results in tighter estimations. Similarly, an optimal allocation of probabilities (configurations of δ_1 and δ_2) can lead to smaller sampling rates and thus higher query speedups. By default, we set $\theta_p = 0.05\%$ and $\delta_1 = \delta_2 = 1 - p' = \frac{1-p}{3}$. In line with existing literature [40, 45, 57, 58, 90], we recommend configuring θ_p to ensure that the pilot

[3]Due to the overhead, sampling is often slower than index seeking, which is often used when the table is indexed and predicates are highly selective.

[4]Sub-Gaussian assumption holds for any bounded distribution based on Hoeffding's inequality. Estimates of aggregate are bounded as tables have finite cardinality.

Table 2. Upper bounds of relative errors of composite estimators with multiplication, division, and addition.

Composite estimator	Upper bound of relative error
$\hat{\mu}_1 \cdot \hat{\mu}_2$	$e_{\mu_1} + e_{\mu_2} + e_{\mu_1} \cdot e_{\mu_1}$
$\hat{\mu}_1 / \hat{\mu}_2$	$(e_{\mu_1} + e_{\mu_2})/(1 + \min(e_{\mu_1}, e_{\mu_2}))$
$\hat{\mu}_1 + \hat{\mu}_2$	$\max(e_{\mu_1}, e_{\mu_2})$

sample typically includes more than 30 units. For those requiring optimal performance, we suggest efficiently tuning δ_1 and δ_1 using cached pilot query results.

Following Procedure 1, we can obtain an estimated aggregate $\hat{\mu}$ that satisfies the user's error specification. We formally state the guarantee in Theorem 3.1.

Theorem 3.1. *Assuming that the aggregate to estimate is sub-Gaussian, if the input query Q_{in} is rewritten into a final query Q_{final} based on the sampling plan Θ obtained from the Procedure 1, the estimated aggregate $\hat{\mu}$ computed in Q_{final} satisfies the inequality: $\mathbb{P}\left[|(\hat{\mu} - \mu)/\mu| \leq e\right] \geq p$.*

Proof Sketch. The probability of relative error bound can be proved by integrating the estimations in the Procedure 1 and the confidence interval for μ using Boole's inequality. We defer the full proof to our technical report [110]. □

In PilotDB, L_μ and $U_V[\Theta]$ cannot be naively obtained through standard techniques since PilotDB uses block sampling, instead of row-level sampling. Block sampling introduces correlations among data from the same block, which breaks the assumption of data independence in standard techniques [5, 43, 57, 79]. We develop a set of novel statistical techniques, BSAP, to address that (§4).

Multi-Aggregate Queries. It is common to calculate more than one aggregate in a single query by computing arithmetic combinations of multiple aggregations, specifying multiple aggregations, or grouping a table by columns. To guarantee the overall error specification on all aggregates, we need to adjust the error requirement (i.e., the relative error e and the confidence p) for each aggregate.

First, we discuss how TAQA deals with composite aggregates that compute (nonlinear) arithmetic combinations of simple aggregates, such as the product of two SUM aggregates. In TAQA, we handle composite aggregates by propagating the relative error of simple aggregates (e.g., the sum aggregates) into the composite aggregates (e.g., the product). In the case of estimating the product of two simple aggregates, the relative error of the product can be bounded above by the relative errors of the factors:

$$\left|\frac{\hat{\mu}_1 \cdot \hat{\mu}_2 - \mu_1 \cdot \mu_2}{\mu_1 \cdot \mu_2}\right| \leq \left|\frac{\hat{\mu}_1 - \mu_1}{\mu_1}\right|\left|\frac{\hat{\mu}_2 - \mu_2}{\mu_2}\right| + \left|\frac{\hat{\mu}_1 - \mu_1}{\mu_1}\right| + \left|\frac{\hat{\mu}_2 - \mu_2}{\mu_2}\right|$$

This inequality shows that it is sufficient to limit the relative error of factors for the relative error of the product to satisfy the error specification. In PilotDB, we allocate the relative error requirement evenly across simple aggregates. Therefore, each simple aggregate will need to satisfy a relative error of $e' = \sqrt{e+1} - 1$.

We refer to this way of using the relative error of simple aggregates to limit the relative error of a composite aggregate as *error propagation*. We introduce propagation rules for multiplication, division, and addition in Table 2, which are inspired by uncertainty propagations [54, 76, 95]. The validity of these rules can be proved with straightforward algebraic transformation. We defer the detailed proof to [110].

Second, in the case where a query computes multiple aggregates, TAQA adjusts the confidence p and applies the procedures in Procedure 1 to each of them. Based on our error semantics (§2.4),

TAQA should guarantee that the joint probability of the relative error of each estimate being less than e is at least p. To analyze the joint probability, we apply Boole's inequality, which decomposes the probability of a union of events into the sum of probabilities of individual events:

$$\mathbb{P}\left[\bigcap_{1\le i\le k,1\le j\le m} \hat{e}_{i,j} \le e\right] = 1 - \mathbb{P}\left[\bigcup_{1\le i\le k,1\le j\le m} \hat{e}_{i,j} \ge e\right] \ge 1 - \sum_{i=1}^{k}\sum_{j=1}^{m} \mathbb{P}\left[\hat{e}_{i,j} \ge e\right]$$

where $\hat{e}_{i,j} = |(\mu_{i,j} - \hat{\mu}_{i,j})/\mu_{i,j}|$ is the relative error of the aggregate estimate $\hat{\mu}_{i,j}$. This inequality shows that it is sufficient to limit the summation of the confidence of individual aggregates for the overall confidence to hold. With such decomposition, we can conveniently allocate the confidence to each aggregate. In PilotDB, we allocate the confidence evenly. Namely, if we have $k \cdot m$ aggregates, each aggregate $\mu_{i,j}$ needs to satisfy its relative error requirement with confidence of $p_{i,j} = 1 - \frac{1-p}{km}$.

Handling Missing Groups. Till now, we have been focusing on analyzing the error of estimations. However, for queries with Group By clauses, it is possible to miss groups in the pilot query due to block sampling. In this case, we may result in a sampling plan that does not guarantee errors of aggregates of missed groups. To address it, TAQA controls the sampling rate of the pilot query to ensure that groups larger than a user-specified value g are not missed with a high probability. If all groups output by the query are smaller than g, TAQA will end up generating a sampling plan with large sampling rates, making the approximate query more expensive than the original query. Such sampling plans will be rejected during the sampling plan optimization (§3.2). Consequently, PilotDB will execute these queries exactly.

To ensure that all groups with size greater than g are included in the pilot query results with a high probability, we propose the following lemma that computes the required sampling rate of the pilot query. We defer the proof of the lemma to [110].

Lemma 3.2. *For a table T with a block size b, block sampling with a sampling rate θ satisfying the condition below ensures that the probability of missing a group of size greater than g is less than p_f.*

$$\theta \ge 1 - \left(1 - \left(1 - p_f\right)^{\lceil g/b\rceil/|T|}\right)^{1/\lceil g/b\rceil} \tag{7}$$

Intuitively, Lemma 3.2 calculates the minimum sampling rate to maintain a high group coverage probability. This result extends the group coverage probability of row-level sampling in prior work (i.e., Proposition 4 of [57]) to block sampling. Empirically, with $g = 200$ and $p_f = 0.05$, no groups are missed for the queries we evaluated (§5.3). Nevertheless, there is an opportunity to integrate block sampling with indexes, such as the outlier index [11], to better support small-group queries; such an extension is left for future work.

3.2 Sampling Plan Optimization

For queries with multiple input tables, Procedure 1 often results in multiple valid sampling plans. TAQA uses optimization methods to find the most efficient plan. We formulate sampling plan optimization as a mathematical optimization problem and derive a solution using cost models.

Problem Formulation. According to Procedure 1, the error specification is satisfied if the sampling plan satisfies each constraint $\phi_{i,j}$ of i-th aggregation and j-th group, as defined below:

$$\phi_{i,j}(\Theta) :\equiv \quad z_{(1+p_{i,j})/2} \cdot \sqrt{U_{V_{i,j}}[\Theta]} \cdot L^{-1}_{\mu_{i,j}} \le e_{i,j}$$

where $p_{i,j}, e_{i,j}$ are the adjusted confidence and the relative error requirement, respectively. The overall constraint $\Phi(\Theta)$ is defined as the conjunction of all individual $\phi_{i,j}(\Theta)$.

However, the sampling plan space defined by $\Phi(\Theta)$ is too broad to locate the most efficient sampling plan quickly. To further narrow down the plan space, we introduce the following additional conditions. First, due to the overhead of sampling, a query with a sampling rate larger than 10% can be as expensive as the exact query (Figure 5). Thus, we only consider sampling plans with sampling rates smaller than 10%, which is consistent with prior work [57]. Second, we only consider sampling plans that minimize the sample rate of one of the tables. Finally, we only sample large tables that are expensive to load, using a similar approach to how we identify tables to sample in the pilot query. We choose tables that will be scanned (not seeked) by the DBMS and are of high cardinality (e.g., fact tables [59]). In our experiment, we set a threshold of 1 million rows. These constraints result in the following space of sampling plans for a query with l large tables.

$$\tilde{\Theta} := \left\{ \arg\min_{\Theta} \theta_i,\ s.t.\ \Phi(\Theta) \wedge D(\Theta, S) \;\middle|\; S \subset \{1, \ldots, l\}, i \in S \right\}$$

where $D(\Theta, S)$ defines the domain of sampling plans:

$$D(\Theta, S) :\equiv \quad (\forall_{i \in S}\ 0 < \theta_i \leq 0.1) \wedge (\forall_{i \notin S}\ \theta_i = 1)$$

In PilotDB, we enumerate the sets of tables to sample and the individual table whose sampling rate we aim to minimize. For each optimization problem, we use the trust region method for fast and robust convergence [10].

Cost-based Optimization. The set of sampling plans $\tilde{\Theta}$ often contain more than one plan. Among them, we must choose the most efficient one to execute. Unfortunately, measuring the exact cost is prohibitively expensive, as it requires executing the plan. Furthermore, cost estimation is a challenging problem, lacking a universal solution for all DBMSs [102]. In PilotDB, we use the cost model of the underlying DBMS to estimate the cost. Most DBMSs offer external APIs to quickly estimate the cost of a query without executing it [51, 63, 83, 85]. For in-memory databases that may not have cost estimators, such as DuckDB [88], we estimate the cost by the volume of scanned data. This is because data scanning can be much more expensive than data processing for in-memory databases [88]. Empirically, the latency to sampling plan optimization is negligible compared to the overall query execution (§5.6).

Furthermore, exact queries are likely to be cheaper to execute than approximate queries with large sampling rates, particularly when small errors are required for queries with high selectivity or large group cardinality. To address it, PilotDB rejects inefficient sampling plans when the estimated cost is larger than that of the exact query. If no sampling plan is feasible, PilotDB will execute the exact queries.

3.3 Query Rewriting

Throughout `TAQA`, we use query rewriting to synthesize and execute intermediate queries on the underlying DBMS. We describe the high-level procedures to rewrite an arbitrary aggregation query into (1) a pilot query Q_{pilot} which computes statistics required by Procedure 1 and (2) a final query Q_{final} which computes the final answer based on the sampling plan optimized in Section 3.2. We demonstrate the query rewriting with an example in Figure 4.

Pilot Query Rewriting. Based on Procedure 1, Q_{pilot} computes different statistics for different sampling methods. For row-level Bernoulli sampling, Q_{pilot} can directly compute aggregates, corresponding standard deviations and the sample size. For block sampling, Q_{pilot} needs to calculate the aggregates and the size for each sampled block. This requires Q_{pilot} to group the result by blocks. We achieve this by specifying the location of physical data blocks as a column expression.[5]

[5]Nearly every DBMS that implements `TABLESAMPLE SYSTEM` supports outputting data locations [30, 48, 49, 84, 86].

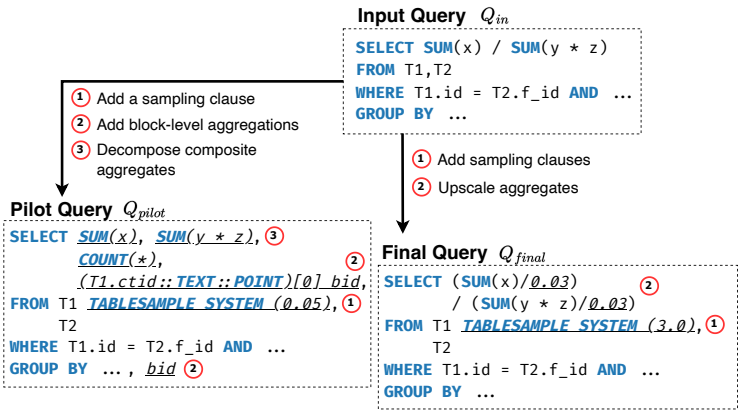


Fig. 4. Demonstration of query rewriting with PostgreSQL syntax. Rewritten parts are emphasized.

For example, in DuckDB, we divide the row ID by the block size; in PostgreSQL, we use the system column `ctid`. We summarize the rewriting procedures as follows:

(1) We add a sampling clause (e.g., `TABLESAMPLE SYSTEM`) to the largest table in Q_{in}.
(2) We incorporate the block location column of the largest table into Group By clauses to compute block-level aggregates.
(3) We decompose composite aggregates (e.g., `SUM(x)/SUM(y)`) into simple aggregates.

Final Query Rewriting. The final query Q_{final} computes estimates of aggregates using the optimized sampling plan obtained. We summarize the rewriting procedures as follows:

(1) We add sampling clauses according to the sampling plan.
(2) We upscale the `SUM`-like aggregates by dividing the product of sampling rates.

4 Block Sampling for Efficient Online AQP

In this section, we address the statistical challenges mentioned in Section 2.1. We first present motivations for using block sampling, examining its benefits and feasibility (§4.1). Next, we develop theoretical results that enable block sampling in AQP with statistical guarantees. That is, we obtain required estimations when using block sampling in Procedure 1 (i.e., L_μ and $U_V[\Theta]$), especially for queries with subqueries (§4.2) and Join (§4.3).

4.1 Motivations

Throughout the history of AQP research, various sampling methods have been studied [2, 5, 27, 57, 72]. However, there is no universally optimal method [13], and block sampling is no exception. Nevertheless, to simultaneously achieve P1, P2, and P3, we argue that block sampling, which samples data blocks, is better than row-level sampling methods. We will explain this from three perspectives that are crucial in choosing sampling methods:

(1) System Efficiency: volume of resulting data in a fixed time
(2) Statistical Efficiency: required sample size for a fixed error rate
(3) Feasibility: achieving statistical guarantees on various DBMSs

System Efficiency. Across sampling methods that do not need DBMS modifications, block sampling achieves the highest system efficiency. This is because block sampling skips non-sampled data.

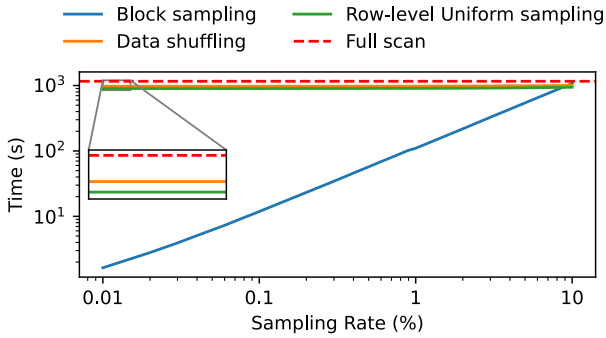


Fig. 5. Comparison of the system efficiency of sampling methods that do not modify DBMSs. At small sampling rates, such as 0.01%, block sampling can be 500× faster than others.

We evaluated the throughput of block sampling, row-level uniform sampling, and data shuffling on a 6B-row table. Figure 5 shows the latency to complete an AVG query over the sampled data with sampling rates from 0.01% to 10% on PostgreSQL. At small sampling rates (e.g., 0.01%), block sampling outperforms others by up to 500×. At large sampling rates (e.g., 10%), all methods have comparable latencies to a full scan.

Statistical Efficiency. Block sampling can achieve comparable or higher statistical efficiency compared to row-level uniform sampling. Intuitively, block sampling introduces correlation across data from the same block, which seems to affect its statistical efficiency. However, in the case when the data of each block is heterogeneous, the statistical efficiency of block sampling can be similar to or better than row-level uniform sampling. We analyze this with an AVG query over a table $\{X_i | 1 \leq i \leq N \cdot b\}$ of N blocks and a consistent block size b.[6] We present the theoretical result in Lemma 4.1 and defer the proof to [110].

Lemma 4.1. *Let σ_j^2 be the variance of data in the j-th block. The ratio between the sample size of block sampling and that of row-level uniform sampling to achieve the same accuracy in expectation is $b\left(1 - \mathbb{E}\left[\sigma_j^2\right] / Var\left[X_i\right]\right)$.*

Based on Lemma 4.1, we analyze the statistical efficiency of block sampling in two cases. First, when each data block is heterogeneous (i.e., $\mathbb{E}\left[\sigma_i^2\right] \to Var[X_{i,j}]$), the required sample size for block sampling can be smaller than that of row-level uniform sampling, achieving better statistical efficiency. Second, when each data block is homogeneous (i.e., $\mathbb{E}\left[\sigma_i^2\right] \to 0$), the required sample size for block sampling is up to b times that of row-level uniform sampling. We found that this rarely happens, especially with deep queries or complex predicates, and is often offset by the system efficiency of block sampling.

Feasibility. Finally, we evaluate whether it is feasible to use block sampling to approximately process arbitrary aggregation queries. We identify two key criteria for this to happen. First, can we obtain unbiased estimations [57]? It is easy to verify that estimations of linear aggregates using block sampling are unbiased. For example, the SUM aggregate can be approximated without bias by adding summations of data blocks divided by the sampling rate. Second, can we achieve statistical guarantees of errors [13]? For queries computing aggregates directly on the output of block sampling, we can achieve error guarantees by analyzing block-level statistics [38, 44, 78]. For

[6]Similarly, we can derive the analysis for *varied* block sizes by treating the block size as a random variable.

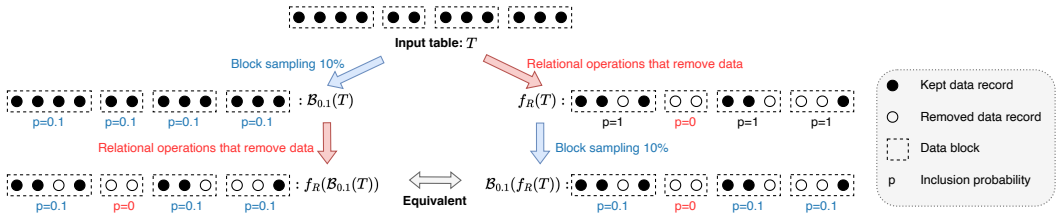


Fig. 6. Demonstration of the commutativity between block sampling $\mathcal{B}$ and relational operations f_R that remove data (e.g., `WHERE`, `JOIN` conditions, and `GROUP BY`); Order of operations does not affect the inclusion probability of each data block.

example, we can obtain a confidence interval of the mean of the sum of each block with standard CLT. However, it is non-trivial to achieve error guarantees for deep nested queries and Join queries. We dedicate the rest of this section to resolving it.

4.2 Deep Nested Queries

Achieving statistical guarantees for sampling-based AQP on deep nested queries is challenging, especially for non-uniform sampling methods [57, 72] such as block sampling. This is because the output of sampling is manipulated by subsequent relational operations, which potentially changes the statistical distribution of the sample. We use the following pair of queries as an example to demonstrate such a situation:

```
-- Q1: the query we execute
SELECT SUM(l_extendedprice * l_discount)
FROM lineitem TABLESAMPLE SYSTEM (0.5%) JOIN parts ON partkey
WHERE l_shipdate >= DATE '1994-01-01' AND ...
-- Q2: the query we analyze
SELECT SUM(l_extendedprice * l_discount)
FROM ( SELECT * FROM lineitem JOIN parts ON partkey
       WHERE l_shipdate >= DATE '1994-01-01' AND ... )
AS cte TABLESAMPLE SYSTEM (0.5%)
```

We can obtain the confidence interval for Q2 by treating the sum of each block as a random variable, similar to prior work [38, 44, 78]. However, it is unclear how to calculate the confidence interval for Q1 due to the Join and filters between block sampling and the aggregation. In this section, we address this issue by analyzing the interaction between block sampling and relational operations and establishing rules for sampling equivalence.

Intuition. In general, we prove that block sampling is commutative with most relational operations, including projection, selection, Join, Group By, and Union. In Figure 6, we demonstrate that exchanging block sampling with any relational operation that removes data does not affect the probability distribution of the sample. For relational operations that add data (e.g., Join), we can always associate added data with a data block where block sampling operates.

Formalization. To formalize and prove this intuition, we define the notion of sampling equivalence in terms of sampling probability.

Definition 4.2. Two sampling procedures, $\mathcal{S}_1$ and $\mathcal{S}_2$, for a set of k relations $\{T_1, \ldots, T_k\}$, where $k \geq 1$, are said to be equivalent, denoted as

$$\mathcal{S}_1(\{T_1, \ldots, T_k\}) \Leftrightarrow \mathcal{S}_2(\{T_1, \ldots, T_k\})$$

if, for any possible sample result R, the probability of obtaining R is the same under both sampling procedures S_1 and S_2, i.e.,

$$\forall R,\ \mathbb{P}\left[\mathcal{S}_1\left(\{T_1, \ldots, T_k\}\right) = R\right] = \mathbb{P}\left[\mathcal{S}_2\left(\{T_1, \ldots, T_k\}\right) = R\right].$$

Next, we derive an important property of the sampling equivalence: the identity of the probability distribution of aggregates, as shown in the following proposition. We defer the proof to [110].

Proposition 4.3. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two equivalent sampling procedures. For any aggregate function f that maps a table to a real value, the probability distribution of the f applied to samples from $\mathcal{S}_1$ is identical to the probability distribution of f applied to the samples from $\mathcal{S}_2$. Namely,*

$$\mathbb{P}\left[f\big(\mathcal{S}_1(\{T_1, \ldots, T_k\})\big) = x\right] = \mathbb{P}\left[f\big(\mathcal{S}_2(\{T_1, \ldots, T_k\})\big) = x\right], \quad \forall x \in \mathbb{R}$$

Based on Proposition 4.3, to show the aggregates computed over the outputs of two different sampling procedures have the same distribution, it is sufficient to prove two sampling procedures are equivalent. Leveraging this, we show that block sampling is commutative with selection, Join, and Union in the following propositions.

Proposition 4.4. *(Selection) For any table T, selection σ_ψ with a predicate ψ, and block sampling $\mathcal{B}_\theta$ with a sampling rate θ,*

$$\sigma_\psi(\mathcal{B}_\theta(T)) \Leftrightarrow \mathcal{B}_\theta(\sigma_\psi(T))$$

Proposition 4.5. *(Join) For any tables T_1 and T_2, Join $\bowtie_\psi$ with a predicate ψ, and block sampling $\mathcal{B}_\theta$ with a sampling rate θ,*

$$\mathcal{B}_\theta(T_1) \bowtie_\psi T_2 \Leftrightarrow \mathcal{B}_\theta(T_1 \bowtie_\psi T_2)$$

Proposition 4.6. *(Union) Let $\cup$ be a bag union operation (or* `UNION ALL` *in SQL). For any tables $T_1, \ldots, T_k$ ($k \geq 2$) and block sampling $\mathcal{B}_\theta$ with a sampling rate θ,*

$$\bigcup_{i=1}^{k} \mathcal{B}_\theta(T_i) \Leftrightarrow \mathcal{B}_\theta\left(\bigcup_{i=1}^{k} T_i\right)$$

Proof Sketch. The derivation of Proposition 4.4, 4.5, and 4.6 closely follows our intuition presented in Figure 6. We defer formal proof to [110]. □

Finally, we consider projection and Group By. We find that the commutativity between block sampling and projection is trivial, since projection is at the column level and thus orthogonal to sampling. Moreover, Group By operations can be considered as a special case of selection with a predicate on the grouping columns.

We conclude these equivalence rules with the following standard form for any supported aggregation query Q:

$$Q \Leftrightarrow \texttt{AGG}\left(\bowtie_{i=1}^{k} \mathcal{B}_{\theta_i}(\tilde{T}_i)\right) \tag{8}$$

where $\tilde{T}_i$ is the output table of intermediate relational operations and θ_i is the sampling rate of the i-th input table. This result is obtained by applying our equivalence rules recursively across the query. Intuitively, if an aggregation query executes block sampling on one input table ($k = 1$), it is equivalent to the query that computes aggregate directly on a block sample. In this case, we can calculate the estimates at the block level and use standard techniques to analyze the error [38]. If a query executes block sampling on multiple input tables ($k > 1$), it is equivalent to the query that computes aggregate on the Join of block samples.

We show that our sampling equivalence rules are stronger than sampling dominance rules of QuickR. First, the sampling dominance rules ensure the validity in only one direction and

do not establish the equivalence. Second, using dominance rules are insufficient for proving the equivalence, as they only consider the inclusion probability of one or two sampled units (i.e., c- and v-dominance). In contrast, our equivalence rules consider the joint inclusion probability of the entire sample. As a result, when two sampling plans are equivalent in our definition, they inherently satisfy sampling dominance.

4.3 Join Queries

When the input query has multiple large tables, TAQA tries to execute block sampling on multiple tables, which leads to Equation 8 with $k > 1$. To analyze the query error with TAQA, we need to (1) ensure Procedure 1 is valid by investigating the asymptotic distribution of the aggregate over the Join of multiple block samples and (2) obtain two estimates L_μ and $U_V[\Theta]$ that are necessary for TAQA to plan sampling (§3.1).

Failure of the Naive Method. Due to correlations within blocks and across Join results, the asymptotic distribution of Equation 8 with $k > 1$ is not governed by standard CLT [14, 38, 47]. Naively applying the standard CLT to calculate confidence intervals can lead to invalid guarantees. We show this failure through the following query that Joins two large tables and uses block sampling on both tables:

```
SELECT SUM(price) FROM lineitem TABLESAMPLE SYSTEM(1%)
INNER JOIN orders TABLESAMPLE SYSTEM(5%)
WHERE l_orderkey = o_orderkey AND comment LIKE '%special%'
```

The "confidence interval" obtained through standard CLT with a 95% intended confidence may have a coverage probability as low as 8%.[7]

Our Solutions. We show that the sample mean still asymptotically converges to a normal distribution when multiple tables of a Join operation are sampled at the block level. However, the variance is not of the standard form. We first present the asymptotic convergence in Theorem 4.7 and defer the proof to [110]. Theorem 4.7 is inspired by [38], but extends their theory to sampling with different rates. We present the theorem in a standard way using the block-level AVG aggregate. The result for SUM and COUNT can be obtained similarly, while the row-level AVG can be considered as a ratio between SUM and COUNT.

Theorem 4.7. *Suppose a Join operation is executed on a set of k tables $\{T_1, \ldots, T_k\}$, where each table T_i has a set of N_i blocks: $\{t_{i,1}, \ldots, t_{i,N_i}\}$. Let $\mathcal{J}(*)$ be a function that takes as input k blocks of different tables and produces the sum of the Join result of these blocks. We denote μ as the block-level mean of the Join result:*

$$\mu = \left(\prod_{i=1}^{k} N_i\right)^{-1} \sum_{i_1=1}^{N_1} \cdots \sum_{i_k=1}^{N_k} \mathcal{J}(t_{1,i_1}, \ldots, t_{k,i_k}) \tag{9}$$

For each Join table T_i, we execute the block sampling with a sample size of n_i blocks. We denote $\hat{\mu}$ as the block-level mean of the Join result of block samples. Then, we can have the following convergence

$$\hat{\mu} - \mu \xrightarrow{D} \mathcal{N}(0, Var[\hat{\mu}]) \quad as \quad n_i \to \infty \tag{10}$$

where $Var[\hat{\mu}]$ is the (unknown) variance of $\hat{\mu}$.

Theorem 4.7 validates our TAQA algorithm on queries where multiple tables are sampled at the block level. To obtain concrete sampling plans, Procedure 1 requires a lower bound of aggregate: L_μ and an upper bound of the variance of the aggregate estimator: $U_V[\Theta]$. We show the results

[7] We evaluated the query on DuckDB with the 1,000-scaled TPC-H 1,000 times.

Table 3. Characteristics of workloads.

Benchmark	#Queries	#Queries w/ Join	Max/Avg. #groups
TPC-H	9	7	175/22
ClickBench	7	0	17/3
SSB	10	10	150/38
Instacart	9	7	146/22
DSB-DBest	169	42	261/52

of $U_V[\Theta]$ for the two-table sampling with a SUM aggregate. L_μ can be derived based on standard probabilistic inequalities, such as Chebyshev's Inequality [43]. We defer the proof to [110].

Lemma 4.8. *Consider a query which Joins two tables T_1 and T_2. Without loss of generality, we suppose that in the pilot query, block sampling with a tiny sampling rate θ_p is executed on T_1, resulting in n_p blocks. Given a final sampling plan $\Theta = [\theta_1, \theta_2]$, the probability that the variance of the* SUM *estimate has an upper bound defined as follows is at least $1 - \delta_2$:*

$$U_V[\Theta] = \frac{1-\theta_1}{\theta_1} U_{y^{(1)}}\left[\frac{\delta_2}{N_2+2}\right] + \frac{1-\theta_2}{\theta_2}\sum_{i_2=1}^{N_2}\left(U_{y_{i_2}^{(2)}}\left[\frac{\delta_2}{N_2+2}\right]\right)^2 + \frac{(1-\theta_1)(1-\theta_2)}{\theta_1\theta_2} U_{y^{(3)}}\left[\frac{\delta_2}{N_2+2}\right]$$

where $y_i^{(1)} = \left(\sum_{i_2=1}^{N_2} \mathcal{J}\left(t_{1,i}, t_{2,i_2}\right)\right)^2$, $y_{i_2,i}^{(2)} = \mathcal{J}\left(t_{1,i}, t_{2,i_2}\right)$, $y^{(3)} = \sum_{i_2=1}^{N_2} \mathcal{J}\left(t_{1,i}, t_{2,i_2}\right)^2$, and $U_y[\delta]$ is the upper bound of the Student's t confidence interval of the summation of y with $1-\delta$ confidence [43].

5 Evaluation

In this section, we evaluate PilotDB with experiments to answer the following questions:

(1) Does PilotDB achieve statistical guarantees (§5.2)?
(2) How much can PilotDB accelerate queries (§5.3)?
(3) How much can BSAP improve existing online AQP (§5.4)?
(4) What are the individual contributions of TAQA and BSAP to overall performance (§5.5)?

5.1 Experiment Settings

Benchmarks. We evaluate PilotDB on a diverse set of benchmarks, including four benchmarks that are widely used in prior work [5, 8, 27, 41, 60, 65, 79] and a benchmark that simulates real-world data with skewed distributions [26]. Other real-world benchmarks used in prior work are proprietary [5, 27], so we cannot evaluate PilotDB on those benchmarks.

- **TPC-H** and **SSB** are synthetic benchmarks for decision-making [23] and star-schema data warehousing [75], respectively. We use a scale factor of 1,000.
- **ClickBench** is a real-world benchmark obtained from the traffic recording of web analytics [20]. We scale up the raw data by 5×, resulting in a pre-processed size of 200 GB.
- **Instacart** is a micro-benchmark with real-world data from the Instacart [52] and queries from TPC-H. We scale up the original data by 100× using the same method as VerdictDB [79].
- **DSB** is a synthetic benchmark based on TPC-DS, blended with skewed yet real-world data distributions, including the (bucketed) exponential distribution and correlations across columns [26]. We use a scale factor of 1,000. To cover the skewness in aggregation, Join, and Group By columns, we use the queries from DBest [66].

In line with previous AQP studies [74, 79], we exclude queries with an empty result, correlated subqueries, and a large group cardinality. In production scenarios, PilotDB can identify those queries via TAQA and execute the exact query. We summarize the key statistics of the workloads in Table 3. A large portion of queries contain Join and various numbers of groups.

DBMSs. We evaluate PilotDB on three DBMSs: PostgreSQL 16.3, SQL Server 2022, and DuckDB 1.0. DuckDB is an open-source in-memory column-oriented DBMS [88]. The default block sampler of DuckDB always scans the entire column, which is less efficient compared to PostgreSQL and SQL Server. To improve the efficiency of DuckDB's block sampling, we add optimization rules to push down block sampling into scanning. Our extension has been merged in DuckDB 1.2 [25].

Baselines. As far as we know, PilotDB is the first AQP system that simultaneously achieves P1, P2, and P3. There are no directly comparable AQP systems to use as a baseline. Hence, we compare PilotDB with executing exact queries on DBMSs that have state-of-the-art query optimizations. In addition, we compare with Quickr [57], the state-of-the-art online AQP system. Quickr achieves P1 and P2 but fails to fulfill P3, which is the closest to PilotDB.

Testbed. Our experiments are conducted on CloudLab [31] r6525 nodes, each equipped with 256 GB RAM, 1.6 TB NVMe SSD, and two 32-core AMD 7543 CPUs. [8] Before executing each query, we clear both the operating system cache and the query plan cache.

5.2 PilotDB Guarantees Errors

We first evaluated whether PilotDB achieves a priori error guarantees. We executed each query from the five benchmarks on PostgreSQL 20 times, each with different target error rates–the maximum relative error in the specification (§2.4). We set the confidence to 95% and measured the maximum relative error of aggregates. By default, we sampled at 0.05% during the planning stage of TAQA. If the input query has Group By clauses, we use Lemma 3.2 with $g = 200, p_f = 0.05$ to compute the sampling rate for planning.

Figures 7 and 7e show the achieved errors for each benchmark with various target errors. The bars in the figure represent the minimum and maximum achieved errors across all queries and executions, while the dots indicate the average achieved errors. For reference, we plot a dashed red line to show the case when the achieved error equals the target error. As shown, PilotDB always achieves errors that are less than the target errors. Furthermore, we find that none of the evaluated queries miss groups, which verifies the effectiveness of Lemma 3.2.

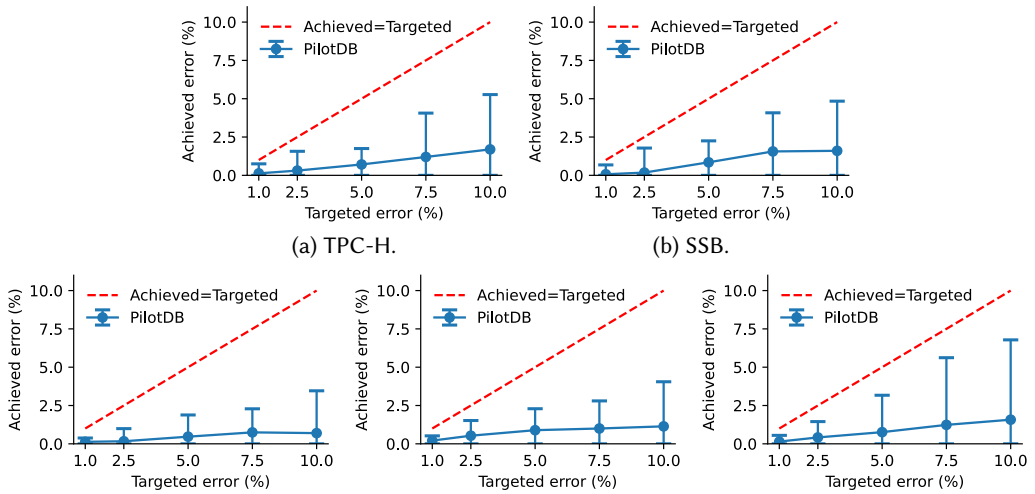
We observe that PilotDB guarantees errors conservatively, with the maximum achieved errors being approximately half of the target errors. This arises because the sampling rates determined by TAQA are guaranteed to be sufficiently large but may not always be the minimum necessary to meet the user's error specifications. For example, we apply Boole's Inequality to tackle the joint probability of multiple events, as shown in Section 3. The equality holds only when events are mutually exclusive. To ensure the sampling rates are also the minimum necessary, it is crucial to analyze the correlations between aggregates, which will be a future work.

We also evaluated the achieved errors when BSAP is replaced with a standard CLT-based confidence interval. We show that without BSAP, the achieved error can be up to 52× higher (1.7× higher on average) than the target error, highlighting the contribution and necessity of BSAP.

5.3 PilotDB Accelerates Query Processing

We analyze the performance of PilotDB by evaluating it on various DBMSs, with different targeted errors, and across all five benchmarks. The query execution follows the setting in Section 5.2.

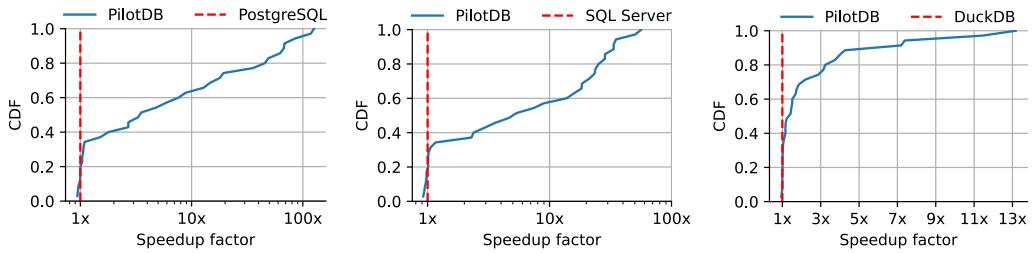
[8] 256 GB RAM is large enough for DuckDB to fit in required columns for individual queries after default compressions.



(a) TPC-H. (b) SSB.

(c) ClickBench. (d) Instacart. (e) DSB.

Fig. 7. PilotDB achieves error guarantees on TPC-H, SSB, ClickBench, Instacart, and DSB. The achieved error is smaller than targeted error if the result is below the red dashed line. We show the maximum, mean, and minimum errors in 20 executions.



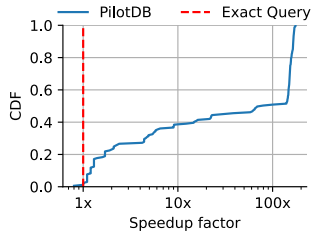
(a) PostgreSQL (log scale). (b) SQL Server (log scale). (c) DuckDB.

Fig. 8. PilotDB achieves 0.92-126× speedups over exact execution across three DBMSs.

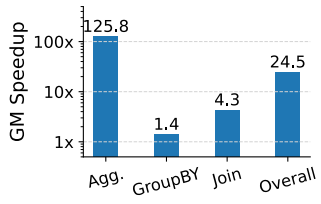
PilotDB Accelerates Queries across Various DBMSs. We evaluated PilotDB on TPC-H, ClickBench, SSB, and Instacart across three DBMSs, targeting a 5% error and 95% confidence. We executed each query in each DBMS 10 times and calculated the geometric mean (GM) of speedups.

Figure 8 provides a detailed view of performance on each database, showing the cumulative probability function (CDF) of speedups compared to exact query execution. As shown, PilotDB consistently accelerates 80% of queries across all DBMSs. Moreover, PilotDB achieves up to 126× speedup on transactional databases and up to 13× speedup on an analytical database, DuckDB. In the worst case, PilotDB slows down the execution by at most 8%. This is because the sample planning stage involves executing a pilot query, the primary overhead causing the loss in performance.

We observe that PilotDB performs better on PostgreSQL and SQL Server than on DuckDB. This is because DuckDB is optimized for in-memory processing. As such, when the data fits in the memory, DuckDB processes queries faster than transactional databases.



(a) Detailed performance on individual queries.



(b) Query speedup grouped by query types.

Fig. 9. PilotDB accelerates queries on skewed data.

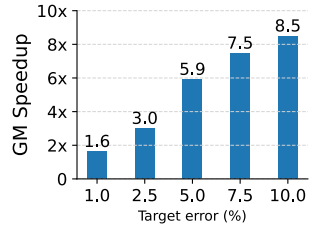


Fig. 10. Speedups of PilotDB across various errors.

PilotDB Accelerates Queries on Skewed Data. To demonstrate the performance of PilotDB on skewed data distributions, we evaluated PilotDB on DSB with a diverse set of 97 aggregation queries, 30 queries with Group By, and 42 queries with Join [66]. We executed each query 10 times on PostgreSQL and calculated the geometric mean of speedups.

Figure 9a shows the CDF of query speedups of PilotDB on DSB. As shown, PilotDB accelerates queries over skewed data by up to two orders of magnitude compared to exact queries on PostgreSQL. To understand how PilotDB performs on different types of queries and skewness, we group query speedups by the query type in Figure 9b. "Agg." refers to simple aggregation queries where the data of aggregated columns is exponentially distributed. "GroupBy" and "Join" refer to queries with exponentially distributed data in the Group By dimension or Join columns, respectively. PilotDB achieves 55× overall speedup and 125× speedup on simple aggregation queries. On Group By and Join queries, PilotDB achieves 1.4× and 4.3× speedup, respectively. This is relatively small compared to simple aggregation queries, but still significant compared to row-level uniform sampling which has 0.9× speedup on average.

PilotDB Accelerates Queries with Various Error Targets. To study how PilotDB performs with different error targets, we evaluated the performance of PilotDB with error targets 1%-10% on PostgreSQL. We executed each query 10 times for each error target and calculated the geometric mean of speedups.

Figure 10 shows the speedup according to different targeted errors. We observe that PilotDB achieves query speedups for all evaluated targeted errors. Even with a small targeted error of 1%, PilotDB achieves 1.6× speedup. As expected, we find that PilotDB achieves higher speedups at larger targeted errors.

Comparison with Quickr. We compared PilotDB with the state-of-the-art online AQP system Quickr. Since Quickr is not open-sourced, we consider a strict performance upper bound of it. Specifically, as mentioned explicitly in their paper [57], Quickr requires one pass over the data. Therefore, we consider the data scanning time on each DBMS as the performance upper bound (i.e., latency lower bound) of Quickr. We give Quickr the benefit of parallelizing scanning with all CPU cores and only consider the elapsed time of the longest scanning operation.

Figure 11 demonstrates the upper bound speedup of Quickr and the speedup of PilotDB across three DBMSs. As shown, PilotDB demonstrates significantly higher query speedup by 1.2-4.2×. Compared to Quickr which always scans the whole data, PilotDB achieves better efficiency by skipping non-sampled data blocks.

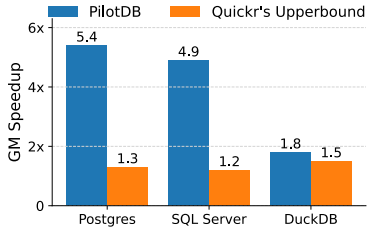


Fig. 11. PilotDB outperforms Quickr by up to 4.2× across three DBMSs.

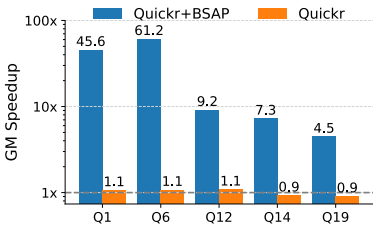


Fig. 12. BSAP improves the speedup of Quickr by up to 60× on DuckDB.

Table 4. Geometric mean of the slowdowns of PilotDB compared to PilotDB-O.

	PostgreSQL	SQL Server	DuckDB
PilotDB (overall)	1.61×	1.21×	1.27×
PilotDB (2nd stage)	1.04×	1.08×	1.19×

5.4 BSAP Augments Existing Online AQP

In this section, we evaluated whether and how much BSAP can improve the performance of existing online AQP. We used TPC-H queries where Quickr applies row-level uniform sampling. On those queries, we reproduce Quickr in DuckDB by manually adopting the rules described in [57] and then rewriting queries with parallelized row-level uniform sampling. We incorporate BSAP into Quickr by further (1) replacing the uniform sampling with block sampling and (2) adapting the Horvitz-Tompson estimator with the error analysis of BSAP. Finally, we target a 10% error, which is consistent with the setting in Quickr's paper [57].

Figure 12 shows the speedups of Quickr+BSAP and original Quickr, compared to exact queries on DuckDB. As shown, Quickr+ BSAP achieves higher speedups by 4.9-60×. We find that these evaluated queries typically have a latency bottleneck at table scanning. In this case, BSAP can significantly accelerate existing online AQP by skipping non-sampled blocks when scanning tables.

5.5 Ablation Study

We evaluated the effectiveness of the design choices of PilotDB by comparing PilotDB with its alternative configurations.

(1) We replace TAQA with pre-computed statistics (PilotDB-O).
(2) We replace BSAP with row-level sampling (PilotDB-R).
(3) We replace Bernoulli sampling with fixed-size sampling.

We used the same setting as Section 5.3 for query executions.

PilotDB Achieves Near-Optimal Performance. In TAQA, we use estimations based on a pilot query to determine the sampling rates for a given error specification (§2.4). To understand the impact of those estimations on the performance of PilotDB, we compare it with PilotDB-O, which represents the upper-bound performance achievable for AQP that uses online block sampling. For each query, we measure the latency of PilotDB-O, PilotDB, and the second stage of PilotDB. We executed all queries in our benchmarks.

Table 4 shows the slowdowns of PilotDB compared to PilotDB-O, computed as the ratio of their latencies. Compared to PilotDB-O, PilotDB is only 21%-61% slower, showing the effectiveness of

Table 5. Speedups of PilotDB over PilotDB-R.

	PostgreSQL	SQL Server	DuckDB
Geometric mean	12.6×	9.37×	1.92×
Maximum	219×	71.4×	13.2×

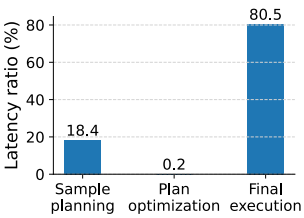


Fig. 13. Latency decomposition of PilotDB.

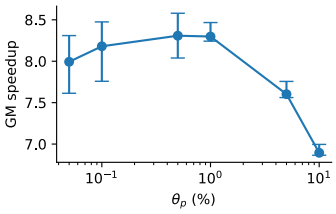


Fig. 14. PilotDB achieves >6× speedup across various θ_p.

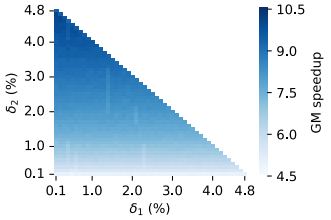


Fig. 15. PilotDB achieves 4.8-10.0× speedups for various (δ_1, δ_2).

TAQA. However, the latency of PilotDB-O does not include the time to determine sampling rates, which requires executing the original input query. To decouple factors that affect the final latency, we also exclude the time to determine the sampling rates in PilotDB, leaving the latency of the second stage of PilotDB. As shown in Table 4, the latency of the second stage of PilotDB is only 4%-19% higher than PilotDB-O. This demonstrates that the optimized sampling plan of PilotDB is close to optimal.

PilotDB Outperforms Row-level Bernoulli Sampling. In Section 4, we showed the advantage of BSAP over uniform row-level sampling with a motivating experiment in Figure 5. Here, we demonstrate the benefit of BSAP in terms of end-to-end latency. We compared PilotDB and PilotDB-R across all the benchmarks. In PilotDB-R, we use the default row-level Bernoulli sampling in each DBMS as the sampling method. That is, we rewrite queries in PostgreSQL and DuckDB with "`TABLESAMPLE BERNOULLI(p)`", where "p" is the sampling rate, and in SQL Server with "`WHERE rand() < p`", where "`rand()`" outputs a random number in [0,1].

Table 5 summarizes the speedup of PilotDB compared to PilotDB-R. We show the geometric mean and maximum speedup for each DBMS. PilotDB achieves a higher geometric mean speedup by 8.0× and a higher maximum speedup by 219×, compared to PilotDB-R. We observe that PilotDB provides a greater benefit on PostgreSQL and SQL Server compared to DuckDB. This is because DuckDB is columnar, which, unlike Postgres and SQL Server, allows it to scan selected columns.

Comparison with Fixed-size Sampling. We compare PilotDB with fixed-size sampling at the row and block level. We use "`ORDER BY RANDOM() LIMIT sample_size`" for row-level fixed-size sampling. Furthermore, only PostgreSQL supports block-level fixed-size sampling, via an extension: `tsm_system_rows` [97]. We repeat both methods on PostgreSQL for TPC-H 10 times, targeting 5% error and 95% confidence. In terms of the geometric mean speedup, PilotDB outperforms row-level fixed-size sampling by 93.3× and underperforms block-level fixed-size sampling by 3.8%. This is because Bernoulli sampling leads to varied sample size which requires sampling more data to maintain the same error guarantees, compared to fixed-size sampling. However, the performance loss is small since the probability of size variation decreases exponentially as the variation amount increases, according to the Chernoff Bound on the Binomial distribution [56].

5.6 Latency Decomposition

We decompose the latency of PilotDB into three parts (1) sample planning (§3.1), (2) plan optimization (§3.2), and (3) final execution. We executed each query on PostgreSQL 10 times and calculated the geometric mean of their latencies. Figure 13 demonstrates the latency proportion of each part. As shown, the sample planning via pilot query execution is the major overhead, while the final query execution constitutes the majority of the total latency.

5.7 Sensitivity Analysis

We conducted a sensitivity analysis of PilotDB's performance across a wide range of parameter settings in Procedure 1: θ_p, δ_1, and δ_2.

Impact of the pilot query sampling rate (θ_p). We executed TPC-H Query 6 on PostgreSQL with various θ_p values (0.05%-10%), aiming for 1% errors and 95% confidence. Figure 14 shows maximum, minimum, and geometric mean speedups achieved by PilotDB across 10 executions. We find that the speedup is non-monotonic with respect to θ_p: performance declines at low sampling rates due to loose estimations and at high rates due to expensive sample planning. Nevertheless, PilotDB achieves >6× speedups consistently.

Impact of the failure probability allocation (δ_1, δ_2). We execute TPC-H query 6 on PostgreSQL with various θ_1 and θ_2 values (0.1%-4.8%) , targeting a 1% error. According to Procedure 1, we ensure $\delta_1 + \delta_2 + p' = 5\%$ to maintain the 95% confidence for the error guarantees. Figure 15 shows the geometric mean speedup of PilotDB across 10 executions. As shown, PilotDB achieves 4.8-10.0× speedups, with the maximum speedup at $\delta_1 = 0.2\%$ and $\delta_2 = 4.6\%$. Our default setting leads to 21% lower speedup compared to the optimal configuration. For scenarios requiring optimal speedups, we can efficiently tune δ_1 and δ_2 with cached pilot query results or incorporate δ_1 and δ_2 as optimizable parameters during the sampling plan optimization.

6 Related Work

Online AQP. Generating samples of large tables upon query arrival is widely studied in prior AQP techniques [32, 57, 72, 73, 103, 106, 107]. Prior work formulated random sampling as a standard operation in query processing to estimate aggregates and used analytical or bootstrap confidence intervals to measure the estimation error [72, 73, 81, 103, 106, 107]. As a step further for complex queries, Quickr injects sampling operations in the query plan level and integrate sample planning with query optimization to achieve acceleration and a priori error guarantees [57]. Additionally, Idea reuses previous results to accelerate future approximate queries [32]. More recently, Taster combines online and offline methods by caching the online samples and reusing them for future queries to achieve faster execution [74].

Although existing online AQP systems return estimation errors, they cannot provide a priori error guarantees without modifying the underlying DBMS. In addition to the DBMS modifications, state-of-the-art methods with a priori error guarantees slow down a significant part of queries compared with exact execution [57, 74] or lead to errors as big as 100% [57].

Offline AQP. Prior work developed two types of offline AQP methods: summary-based methods [16, 34, 35, 71, 77, 80, 82, 92] and sampling-based method [1–5, 7, 11, 12, 27, 33, 65, 74, 79]. The primary idea of summary-based offline AQP is to compress or summarize columns through numeric transformations. Therefore, they cannot process queries with non-numeric columns, such as categorical columns, or with complex relational operations, such as join and grouping.

Sampling-based offline AQP generate offline samples to answer online queries. Aqua first developed the method of rewriting queries with pre-computed samples to answer approximate queries

[1–4]. Subsequently, various optimizations in offline sample creation have been proposed, such as weighted sampling [7, 33], stratified sampling [12], and outlier index [11]. Prior work has explored guaranteeing errors a priori by generating specialized samples for non-nested queries [65], sparse data distribution [104], queries over specific columns [5], and queries with specific selectivities [27]. Furthermore, VerdictDB proposed developing offline AQP as a middleware to avoid modifications to DBMSs [79].

Moreover, sampling-based offline AQP methods have two fundamental limitations. First, their a priori error guarantees are inherently limited to predictable workloads [5, 27, 65, 104]. For example, BlinkDB requires that incoming queries only access columns in a pre-defined column set; Sample+Seek relies on the prior knowledge of the query selectivity to select the right processing policy (i.e., sample or seek). Moreover, maintaining offline samples requires special effort and costs, including regularly refreshing samples to ensure statistical correctness and regenerating samples when the database changes [1, 5, 79].

Online Aggregation. Previous research has explored interactive processing of aggregation queries, providing initial results immediately and improving accuracy as more data is sampled [9, 28, 36, 39, 53, 93, 101, 105]. OLA, first proposed by Hellerstein et al. [39], has been subsequently improved to support join queries [36, 64], scalable processing on large databases [28, 53], processing multiple queries simultaneously [101], and complex aggregates [105]. Furthermore, ProgressiveDB explored online aggregation as an extension to existing DBMSs using progressive views [9]. More recently, DeepOLA tackled nested queries for online aggregation [93].

Although OLA techniques can continuously update confidence intervals, it is invalid to consider the monitored confidence interval as an error guarantee due to the problem of peeking at early results [55]. Nevertheless, OLA can be integrated with the second stage of PilotDB to provide constantly updating results, thereby improving the interactivity and user experience.

Block Sampling. In block sampling, data is sampled at the level of physical data blocks or pages, a method widely recognized as a more efficient sampling scheme than row-level sampling [17–19, 37, 44, 89]. Prior work has studied confidence intervals for aggregates computed directly over the output of block sampling [17, 44, 78], block sampling mixed with row-level sampling (i.e., bi-level sampling) [18, 37], and improved the statistical efficiency of block sampling with block-level summary statistics [89]. However, statistical guarantees for complex approximate queries (e.g., nested queries and Join queries) with block sampling have not been investigated in literature.

7 Conclusion

We propose PilotDB, an online AQP system that achieves (1) a priori error guarantees, (2) no maintenance overheads, and (3) no DBMS modifications. To achieve these properties, we propose a novel online AQP algorithm, TAQA, based on query rewriting and online sampling. To accelerate queries with TAQA, we formalize block sampling with new statistical techniques to provide guarantees on nested queries and Join queries. Our evaluation shows that PilotDB consistently achieves a priori error guarantees and accelerates queries by up to 126× on various DBMSs.

Acknowledgments

This work was supported in part by NSF under grant CCF-2316233. We are grateful to the CloudLab for providing computing resources for experiments [31]. We thank Andy Luo, Chuxuan Hu, and Lilia Tang for their feedback and help.

References

[1] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. 1999. Aqua: A fast decision support systems using approximate query answers. In *PVLDB*.

[2] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. 2000. Congressional samples for approximate answering of group-by queries. In *SIGMOD*.

[3] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The aqua approximate query answering system. In *SIGMOD*.

[4] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join synopses for approximate query answering. In *SIGMOD*.

[5] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*.

[6] Azure Synapse Analytics. [n. d.]. *Designing tables*. https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-overview Accessed: 2024-05-12.

[7] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic sample selection for approximate query processing. In *SIGMOD*.

[8] Leilani Battle, Philipp Eichmann, Marco Angelini, Tiziana Catarci, Giuseppe Santucci, Yukun Zheng, Carsten Binnig, Jean-Daniel Fekete, and Dominik Moritz. 2020. Database benchmarking for supporting real-time interactive querying of large data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1571–1587.

[9] Lukas Berg, Tobias Ziegler, Carsten Binnig, and Uwe Röhm. 2019. ProgressiveDB: progressive data analytics as a middleware. *PVLDB* (2019).

[10] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. 1999. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization* 9, 4 (1999), 877–900.

[11] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayya. 2001. Overcoming limitations of sampling for aggregation queries. In *ICDE*.

[12] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems* (2007).

[13] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 511–519.

[14] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On random sampling over joins. *ACM SIGMOD Record* 28, 2 (1999), 263–274.

[15] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1401–1412.

[16] Jiecao Chen and Qin Zhang. 2017. Bias-Aware Sketches. *PVLDB* (2017).

[17] Xingguang Chen, Fangyuan Zhang, and Sibo Wang. 2022. Efficient Approximate Algorithms for Empirical Variance with Hashed Block Sampling. In *KDD*.

[18] Yu Cheng, Weijie Zhao, and Florin Rusu. 2017. Bi-level online aggregation on raw data. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 1–12.

[19] Xiang Ci and Xiaofeng Meng. 2015. An efficient block sampling strategy for online aggregation in the cloud. In *Web-Age Information Management: 16th International Conference*. Springer.

[20] ClickHouse. [n. d.]. *ClickBench: a Benchmark For Analytical Databases*. https://github.com/ClickHouse/ClickBench Accessed: 2024-06-04.

[21] Google Cloud. [n. d.]. *Approximate aggregate functions | BigQuery*. https://cloud.google.com/bigquery/docs/reference/standard-sql/approximate_aggregate_functions Accessed: 2024-06-05.

[22] Google Cloud. [n. d.]. *Table sampling | BigQuery | Google Cloud*. https://cloud.google.com/bigquery/docs/table-sampling Accessed: 2024-05-12.

[23] Transaction Processing Performance Council. [n. d.]. *TPC-H Homepage*. https://www.tpc.org/tpch/ Accessed: 2024-06-04.

[24] Azure Databricks. 2024. *TABLESAMPLE clause*. https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-syntax-qry-select-sampling Accessed: 2024-05-12.

[25] DuckDB Developers. 2025. DuckDB 1.2.0 "Histrionicus". https://github.com/duckdb/duckdb/releases/tag/v1.2.0

[26] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. 2021. DSB: A decision support benchmark for workload-driven and traditional database systems. *Proceedings of the VLDB Endowment* 14, 13 (2021), 3376–3388.

[27] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. 2016. Sample+ seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*.

[28] Alin Dobra, Chris Jermaine, Florin Rusu, and Fei Xu. 2009. Turbo-charging estimate convergence in DBO. *PVLDB* (2009).

[29] DuckDB. [n. d.]. *Samples*. https://duckdb.org/docs/sql/samples.html Accessed: 2024-05-12.

[30] DuckDB. [n. d.]. *SELECT Statement*. https://duckdb.org/docs/sql/statements/select.html#row-ids Accessed: 2024-06-21.

[31] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[32] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2017. Revisiting reuse for approximate query processing. *PVLDB* (2017).

[33] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. 2000. Icicles: Self-tuning samples for approximate query answering. In *PVLDB*.

[34] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2002. Querying and Mining Data Streams: You Only Get One Look. (2002).

[35] Sudipto Guha and Boulos Harb. 2005. Wavelet synopsis for data streams: minimizing non-euclidean error. In *KDD*.

[36] Peter J Haas and Joseph M Hellerstein. 1999. Ripple joins for online aggregation. *ACM SIGMOD Record* (1999).

[37] Peter J Haas and Christian König. 2004. A bi-level bernoulli scheme for database sampling. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 275–286.

[38] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Arun N Swami. 1996. Selectivity and cost estimation for joins based on random sampling. *J. Comput. System Sci.* 52, 3 (1996), 550–569.

[39] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *SIGMOD*.

[40] Melody A Hertzog. 2008. Considerations in determining sample size for pilot studies. *Research in nursing & health* 31, 2 (2008), 180–191.

[41] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. 2020. Learning a partitioning advisor for cloud databases. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 143–157.

[42] Apache Hive. [n. d.]. *LanguageManual Sampling*. https://cwiki.apache.org/confluence/display/hive/languagemanual+sampling Accessed: 2024-05-12.

[43] Robert V Hogg, Joseph W McKean, and Allen T Craig. 2019. *Introduction to mathematical statistics*. Pearson.

[44] Wen-Chi Hou and Gultekin Ozsoyoglu. 1991. Statistical estimators for aggregate relational algebra queries. *ACM Transactions on Database Systems (TODS)* (1991).

[45] Wen-Chi Hou, Gultekin Ozsoyoglu, and Erdogan Dogdu. 1991. Error-constrained COUNT query evaluation in relational databases. *ACM SIGMOD Record* 20, 2 (1991), 278–287.

[46] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K Taneja. 1988. Statistical estimators for relational algebra expressions. In *SIGMOD*.

[47] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. 2019. Joins on samples: A theoretical guide for practitioners. *arXiv preprint arXiv:1912.03443* (2019).

[48] Microsoft Community Hub. [n. d.]. *Where is a record really located?* https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/where-is-a-record-really-located/ba-p/370972 Accessed: 2024-06-21.

[49] Apache Impala. [n. d.]. *Impala 4.0.0 Documentation: Table and Column Statistics*. https://docs.cloudera.com/cdw-runtime/cloud/impala-sql-reference/topics/impala-virtual-columns.html#pnavId1 Accessed: 2024-06-21.

[50] Apache Impala. [n. d.]. *TABLESAMPLE Clause*. https://impala.apache.org/docs/build/html/topics/impala_tablesample.html Accessed: 2024-05-12.

[51] Apache Impala. [n. d.]. *Understanding Impala Query Performance*. https://impala.apache.org/docs/build/html/topics/impala_explain_plan.html Accessed: 2024-09-21.

[52] Instacart. [n. d.]. *Instacart Market Basket Analysis*. https://www.kaggle.com/c/instacart-market-basket-analysis/data Accessed: 2024-05-12.

[53] Chris Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. 2008. Scalable approximate query processing with the DBO engine. *ACM Transactions on Database Systems* (2008).

[54] Clemens Jochum, Peter Jochum, and Bruce R Kowalski. 1981. Error propagation and optimal performance in multicomponent analysis. *Analytical Chemistry* 53, 1 (1981), 85–92.

[55] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. 2017. Peeking at a/b tests: Why it matters, and what to do about it. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1517–1525.

[56] N Singh Kambo and Samuel Kotz. 1966. On exponential bounds for binomial probabilities. *Annals of the Institute of Statistical Mathematics* 18, 1 (1966), 277–287.

[57] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*.

[58] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating approximate aggregation queries with expensive predicates. *arXiv preprint arXiv:2108.06313* (2021).
[59] Ralph Kimball and Margy Ross. 2013. *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons.
[60] Andrew Lamb, Yijie Shen, Daniël Heres, Jayjeet Chakraborty, Mehmet Ozan Kabak, Liang-Chi Hsieh, and Chao Sun. 2024. Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine. In *Companion of the 2024 International Conference on Management of Data*. 5–17.
[61] Database Languages. 2003. SQL, ISO/IEC 9075*:2003.
[62] Microsoft Learn. [n. d.]. *Intelligent query processing features in detail*. https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver16#approximate-query-processing Accessed: 2024-06-05.
[63] Microsoft Learn. [n. d.]. *SET SHOWPLAN_ALL (Transact-SQL)*. https://learn.microsoft.com/en-us/sql/t-sql/statements/set-showplan-all-transact-sql?view=sql-server-ver16 Accessed: 2024-09-21.
[64] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander join: Online aggregation via random walks. In *SIGMOD*.
[65] Kaiyu Li, Yong Zhang, Guoliang Li, Wenbo Tao, and Ying Yan. 2018. Bounded approximate query processing. *TKDE* (2018).
[66] Qingzhi Ma and Peter Triantafillou. 2019. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*. 1553–1570.
[67] Microsoft. 2024. *SQL Server 2022 CU13*. https://packages.microsoft.com/ubuntu/20.04/mssql-server-2022/pool/main/m/mssql-server/
[68] Barzan Mozafari. 2017. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*.
[69] Barzan Mozafari, Radu Alexandru Burcuta, Alan Cabrera, Andrei Constantin, Derek Francis, David Grömling, Alekh Jindal, Maciej Konkolowicz, Valentin Marian Spac, Yongjoo Park, et al. 2023. Making Data Clouds Smarter at Keebo: Automated Warehouse Optimization using Data Learning. In *Companion of the 2023 International Conference on Management of Data*. 239–251.
[70] Barzan Mozafari, Eugene Zhen Ye Goh, and Dong Young Yoon. 2015. Cliffguard: A principled framework for finding robust database designs. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1167–1182.
[71] Ioannis Mytilinis, Dimitrios Tsoumakos, and Nectarios Koziris. 2016. Distributed wavelet thresholding for maximum error metrics. In *Proceedings of the 2016 International Conference on Management of Data*. 663–677.
[72] Supriya Nirkhiwale, Alin Dobra, and Christopher Jermaine. 2013. A Sampling Algebra for Aggregate Estimation. *PVLDB* (2013).
[73] Frank Olken and Doron Rotem. 1986. Simple Random Sampling from Relational Databases. In *PVLDB*.
[74] Matthaios Olma, Odysseas Papapetrou, Raja Appuswamy, and Anastasia Ailamaki. 2019. Taster: Self-tuning, elastic and online approximate query processing. In *ICDE*. IEEE.
[75] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. 2007. The star schema benchmark (SSB). *Pat* 200, 0 (2007), 50.
[76] M Palmer. 2003. Propagation of uncertainty through mathematical operations. *Massachusetts Institute of* (2003).
[77] Prashant Pandey, Michael A Bender, Rob Johnson, and Rob Patro. 2017. A general-purpose counting filter: Making every bit count. In *SIGMOD*.
[78] Niketan Pansare, Vinayak Borkar, Chris Jermaine, and Tyson Condie. 2011. Online aggregation for large mapreduce jobs. *Proceedings of the VLDB Endowment* 4, 11 (2011), 1135–1145.
[79] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. Verdictdb: Universalizing approximate query processing. In *SIGMOD*.
[80] Gregory Piatetsky-Shapiro and Charles Connell. 1984. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record* (1984).
[81] Abhijit Pol and Christopher Jermaine. 2005. Relational confidence bounds are easy with the bootstrap. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 587–598.
[82] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD Record* (1996).
[83] PostgreSQL. [n. d.]. *EXPLAIN*. https://www.postgresql.org/docs/current/sql-explain.html Accessed: 2024-09-21.
[84] PostgreSQL. [n. d.]. *PostgreSQL: Documentation: 16: 5.5. System Columns*. https://www.postgresql.org/docs/current/ddl-system-columns.html#DDL-SYSTEM-COLUMNS-CTID Accessed: 2024-06-21.
[85] Presto. [n. d.]. *Cost in Explain*. https://prestodb.io/docs/current/optimizer/cost-in-explain.html Accessed: 2024-09-21.
[86] Presto. [n. d.]. *Hive Connector*. https://prestodb.io/docs/current/connector/hive.html#extra-hidden-columns Accessed: 2024-06-21.

[87] Presto. [n. d.]. *SELECT — Presto 0.287 Documentation.* https://prestodb.io/docs/current/sql/select.html#tablesample Accessed: 2024-05-12.
[88] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*. 1981–1984.
[89] Kexin Rong, Yao Lu, Peter Bailis, Srikanth Kandula, and Philip Levis. 2020. Approximate partition selection for big-data workloads using summary statistics. *PVLDB* (2020).
[90] Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. 2023. Accelerating aggregation queries on unstructured streams of data. *arXiv preprint arXiv:2308.09157* (2023).
[91] Praveen Seshadri, Hamid Pirahesh, and TY Cliff Leung. 1996. Complex query decorrelation. In *Proceedings of the Twelfth International Conference on Data Engineering*. IEEE, 450–458.
[92] Michael Shekelyan, Anton Dignös, and Johann Gamper. 2017. Digithist: a histogram-based data summary with tight error bounds. *PVLDB* (2017).
[93] Nikhil Sheoran. 2022. Deepola: Online aggregation for deeply nested queries. In *SIGMOD*.
[94] Hong Su, Mohamed Zait, Vladimir Barrière, Joseph Torres, and Andre Menck. 2016. Approximate aggregates in oracle 12c. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1603–1612.
[95] Talia Tamarin-Brodsky and John Marshall. [n. d.]. Error Analysis. ([n. d.]).
[96] PostgreSQL Team. 2024. *PostgreSQL 16.3*. The PostgreSQL Global Development Group.
[97] PostgreSQL Team. 2025. tsm_system_rows — the SYSTEM_ROWS sampling method for TABLESAMPLE. https://www.postgresql.org/docs/current/tsm-system-rows.html
[98] Dimitri Theodoratos, Timos Sellis, et al. 1997. Data warehouse configuration. In *VLDB*, Vol. 97. 126–135.
[99] Congying Wang, Nithin Sastry Tellapuri, Sphoorthi Keshannagari, Dylan Zinsley, Zhuoyue Zhao, and Dong Xie. 2023. Approximate Queries over Concurrent Updates. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3986–3989.
[100] PostgreSQL wiki. [n. d.]. *TABLESAMPLE Implementation.* https://wiki.postgresql.org/wiki/TABLESAMPLE_Implementation Accessed: 2024-05-12.
[101] Sai Wu, Beng Chin Ooi, and Kian-Lee Tan. 2010. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*.
[102] Wentao Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigümüs, and Jeffrey F Naughton. 2013. Predicting query execution time: Are optimizer cost models really unusable?. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 1081–1092.
[103] Fei Xu, Christopher Jermaine, and Alin Dobra. 2008. Confidence bounds for sampling-based group by estimates. *ACM Transactions on Database Systems* (2008).
[104] Ying Yan, Liang Jeff Chen, and Zheng Zhang. 2014. Error-bounded sampling for analytics on big sparse data. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1508–1519.
[105] Kai Zeng, Sameer Agarwal, Ankur Dave, Michael Armbrust, and Ion Stoica. 2015. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*.
[106] Kai Zeng, Shi Gao, Jiaqi Gu, Barzan Mozafari, and Carlo Zaniolo. 2014. ABS: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*.
[107] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. 2014. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*. 277–288.
[108] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data*. 1525–1539.
[109] Zhuoyue Zhao, Dong Xie, and Feifei Li. 2022. AB-tree: index for concurrent random sampling and updates. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1835–1847.
[110] Yuxuan Zhu, Tengjun Jin, Stefanos Baziotis, Chengsong Zhang, Charith Mendis, and Daniel Kang. 2025. *PilotDB: Database-Agnostic Online Approximate Query Processing with A Priori Error Guarantees (Technical Report).* https://arxiv.org/abs/2503.21087

Received October 2024; revised January 2025; accepted February 2025