DProvDB: Differentially Private Query Processing with Multi-Analyst Provenance

Anonymous Author(s)

ABSTRACT

Recent years have witnessed the adoption of differential privacy (DP) in practical database systems like PINQ, Flex, and PrivateSQL. Such systems allow data analysts to query sensitive data while providing a rigorous and provable privacy guarantee. However, the existing design of these systems does not distinguish data analysts of different privilege levels or trust levels. This design can have an unfair apportion of the privacy budget among the data analyst if treating them as a single entity, or waste the privacy budget if considering them as non-colluding parties and answering their queries independently. In this paper, we propose *DProvDB*, a fine-grained privacy provenance framework for multi-analysts that tracks the privacy loss to each single data analyst. Under this framework, when given a fixed privacy budget, we build algorithms that maximize the number of queries that could be answered accurately and apportion the privacy budget according to the privilege level of the data analysts.

1 INTRODUCTION

With the growing attention on data privacy and the development of privacy regulations like GDPR [52], companies with sensitive data must share their data without compromising the privacy of data contributors. Differential privacy (DP) [21] has been considered as a promising standard for this setting. Recent years have witnessed the adoption of DP to practical systems for data management and online query processing, such as PINQ [40], FLEX [29], PrivateSQL [35], GoogleDP [1], and Chorus [28]. In systems of this kind, data curators or system providers set up a finite system-wise privacy budget to bound the overall extent of information disclosure. An incoming query consumes some privacy budget. The system stops processing new queries once the budget has been fully depleted. Thus, the privacy budget is a crucial resource to manage in such a query processing system.

In practice, multiple data analysts can be interested in the same data, and they have different privilege/trust levels in accessing the data. For instance, tech companies need to query their users' data for internal applications like anomaly detection. They also consider inviting external researchers with low privilege/trust levels to access the same sensitive data for study. Existing query processing systems with DP guarantees would regard these data analysts as a unified entity and do not provide tools to distinguish them or track their perspective privacy loss. This leads to a few potential problems. First, a low-privilege external data analyst who asks queries first can consume more privacy budget than an internal one, if the system does not interfere with the sequence of queries. Second, if naïvely tracking and answering each analyst's queries independently of the others, the system can waste the privacy budget when two data analysts ask similar queries.

The aforementioned challenges to private data management and analytics are mainly on account of the fact that the systems are "*stateless*", meaning none of the existing DP query-processing systems records the individual budget limit and the historical queries asked by the data analysts. That is, the **metadata** about *where the query comes from, how the query is computed, and how many times each result is produced*, which is related to the **provenance information** in database research [8, 11]. As one can see, without privacy provenance, the query answering process for the multi-analyst use case can be unfair or wasteful in budget allocation.

To tackle these challenges, we propose a "stateful" DP query processing system *DProvDB*, which enables a novel privacy provenance framework designed for the multi-analysts setting. Following existing work [36], *DProvDB* answers queries based on private synopses (i.e., materialized results for views) of data. Instead of recording all the query metadata, we propose a more succinct data structure, a privacy provenance table, that enforces only necessary privacy tracing as per each data analyst and per view. The privacy provenance table is associated with privacy constraints so that constraint-violating queries will be rejected. Making use of this privacy provenance framework, *DProvDB* can maintain global (viz., as per view) and local (viz., as per analyst) DP synopses and update them dynamically according to data analysts' requests.

DProvDB is also supported with a new DP mechanism, called *additive Gaussian mechanism*. This mechanism first creates a global DP synopsis for a view query; Then, from this global synopsis, it provides the necessary local DP synopsis to data analysts who are interested in this view by adding more Gaussian noise. In such a way *DProvDB* is tuned to answer as many queries accurately from different data analysts. Even when all the analysts collude, the privacy loss will be bounded by the budget used for the global synopsis. Adding up to its merits, we notice the provenance tracking in *DProvDB* can help in achieving a notion called proportional fairness. We believe most of if not all, existing DP query processing systems can benefit from integrating our multi-analyst privacy provenance framework — *DProvDB* can be regarded as a middle-ground approach between the purely interactive DP systems and those based solely on synopses, from which we both provably and empirically show that *DProvDB* can significantly improve on system utility and fairness for multi-analyst DP query processing.

The contributions of this paper are the following:

- We propose a multi-analyst DP model where mechanisms satisfying multi-analyst DP provide discrepant answers to analysts with different privilege levels. Under this setting, we ask research questions about tight privacy analysis, budget allocation, and fair query processing. (Section §3)
- We propose a privacy provenance framework that compactly traces historical queries and privacy consumption as per analyst and as per view. With this framework, the administrator is able to enforce privacy constraints, enabling dynamic budget allocation and fair query processing. (Section §4)

Table 1: Notation Summary

Notation	*Definition*
$\epsilon(, \delta)$	The privacy budget
$\mathcal{A}, A_i$	(A set of) Data analyst(s)
$\mathcal{D}, D$	Database domain, database instance
$\mathcal{P}$	The privacy provenance table
Ψ, ψ	(A set of) Privacy constraint(s)
$\mathcal{V}, V$	(A set of) View(s)
$V^{\epsilon}, V^{\epsilon}_{A_i}$	Global/Local synopsis

- We design new accuracy-aware DP mechanisms that leverages the provenance information to manage synopses and inject correlated noise to achieve tight collusion bound in the multi-analyst setting. The proposed mechanisms can be seamlessly added to the algorithmic toolbox for DP systems. (Section §5)
- We implement *DProvDB*[1], as a new multi-analyst query processing interface, and integrate it into an existing DP query system. We empirically evaluate *DProvDB*, and the experimental results show that our system is efficient and effective compared to baseline systems. (Section §7)

Paper Roadmap. The remainder of this paper is outlined as follows. Section 2 introduces the necessary notations and background knowledge on database and differential privacy. Our multi-analysts DP query processing research problems are formulated in section 3 and a high-level overview of our proposed system is briefed in section 4. Section 5 describes the details of our design of the DP mechanisms and system modules. Section 6 discusses extensions to the collusion model, fairness, and approaches to answering highly sensitive queries. In section 7, we present the implementation details and an empirical evaluation of our system against the baseline solutions. In section 8 we go through the related works and we conclude this work in section 9.

2 PRELIMINARIES

Let $\mathcal{D}$ denotes the domain of database and D be a database instance. A relation $R \in D$ consists of a set of attributes, $attr(R) = \{a_1, \ldots, a_j\}$. We denote the domain of an attribute a_j by $Dom(a_j)$ while $|Dom(a_j)|$ denotes the domain size of that attribute. We introduce and summarize the related definitions of differential privacy.

Definition 1 (Differential Privacy [20]). *We say that a randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{O}$ satisfies (ϵ, δ)-differential privacy (DP), if for any two neighbouring databases (D, D') that differ in only 1 tuple, and $O \subseteq \mathcal{O}$, we have*

$$\Pr[\mathcal{M}(D) \in O] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in O] + \delta.$$

DP enjoys many useful properties, for example, post-processing and sequential composition [21].

Theorem 2.1 (Sequential Composition [21]). *Given two mechanisms $\mathcal{M}_1 : \mathcal{D} \to \mathcal{O}_1$ and $\mathcal{M}_2 : \mathcal{D} \to \mathcal{O}_2$, such that $\mathcal{M}_1$ satisfies (ϵ_1, δ_1)-DP and $\mathcal{M}_2$ satisfies (ϵ_2, δ_2)-DP. The combination of the two mechanisms $\mathcal{M}_{1,2} : \mathcal{D} \to \mathcal{O}_1 \times \mathcal{O}_2$, which is a mapping $\mathcal{M}_{1,2}(D) = (\mathcal{M}_1(D), \mathcal{M}_2(D))$, is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP.*

[1]The system code is available at https://anonymous.4open.science/r/DProvDB-665C.

Theorem 2.2 (Post Processing [21]). *For an (ϵ, δ)-DP mechanism $\mathcal{M}$, applying any arbitrary function f over the output of $\mathcal{M}$, that is, the composed mechanism $f \circ \mathcal{M}$, satisfies (ϵ, δ)-DP.*

The sequential composition bounds the privacy loss of the sequential execution of DP mechanisms over the database. It can naturally be generalized to the composition of k differentially private mechanisms. The post-processing property of DP indicates that the execution of any function on the output of a DP mechanism will not incur privacy loss.

Definition 2 (ℓ_2-Global Sensitivity). *For a query $q : \mathcal{D} \to \mathbb{R}^d$ the ℓ_2 global sensitivity of this query is*

$$\Delta q = \max_{D, D' : d(D, D') \leq 1} \|q(D) - q(D')\|_2,$$

where $d(\cdot, \cdot)$ denotes the number of tuples that D and D' differ and $\|\cdot\|_2$ denotes the ℓ_2 norm.

Definition 3 (Analytic Gaussian Mechanism [2]). *Given a query $q : \mathcal{D} \to \mathbb{R}^d$, the analytic Gaussian mechanism $\mathcal{M}(D) = q(D) + \eta$ where $\eta \sim \mathcal{N}\left(0, \sigma^2 I\right)$ is (ϵ, δ)-DP if and only if*

$$\Phi_{\mathcal{N}}\left(\frac{\Delta q}{2\sigma} - \frac{\epsilon\sigma}{\Delta q}\right) - e^{\epsilon}\Phi_{\mathcal{N}}\left(-\frac{\Delta q}{2\sigma} - \frac{\epsilon\sigma}{\Delta q}\right) \leq \delta,$$

where $\Phi_{\mathcal{N}}$ denotes the cumulative density function (CDF) of Gaussian distribution.

In this mechanism, the Gaussian variance is determined by $\sigma = \alpha\Delta q/\sqrt{2\epsilon}$ where α is a parameter determined by ϵ and δ [2].

DP mechanisms involve errors in the query answer. A common utility function for numerical query answers is the expected mean squared error (MSE), defined as follows:

Definition 4 (Query Utility). *For a query $q : \mathcal{D} \to \mathbb{R}^d$ and a mechanism $\mathcal{M} : \mathcal{D} \to \mathbb{R}^d$, the query utility of mechanism $\mathcal{M}$ is measured as the expected squared error, $v = \mathbb{E}[q(D) - \mathcal{M}(D)]^2 = \sum_{i=1}^{n}(q(D)[i] - \mathcal{M}(D)[i])^2$. For the (analytic) Gaussian mechanism, the expected squared error equals its variance, that is, $v = d\sigma^2$.*

3 PROBLEM SETUP

We consider the **multi-analysts** setting, where there are multiple data analysts $\mathcal{A} = \{A_1, \ldots, A_m\}$ who want to ask queries on the database D. The administrator who manages the database wants to ensure that the sensitive data is properly and privately shared with the data analysts $A_1, \ldots, A_m$. In our threat model, the data analysts can adaptively select and submit arbitrary queries to the system to infer sensitive information about individuals in the protected database. Additionally, in our multi-analysts model, data analysts may submit the same query and collude to learn more information about the sensitive data.

In practice, data analysts have varying privilege levels, which the standard DP does not consider. An analyst with a higher privilege level is allowed more information than an analyst with a lower privilege level. This motivates us to present the following variant of DP that supports the privacy per analyst provenance framework and guarantees different levels of privacy loss to the analysts.

Definition 5 (Multi-analyst DP). *We say a randomized mechanism $\mathcal{M} : \mathcal{D} \to (O_1, \ldots, O_m)$ satisfies $[(A_1, \epsilon_1, \delta_1), ..., (A_m, \epsilon_m, \delta_m)]$-multi-analyst-DP if for any neighbouring databases (D, D'), any*

$i \in [m]$, and all $O_i \subseteq O_i$, we have

$$\Pr[\mathcal{M}(D) \in O_i] \leq e^{\epsilon_i} \Pr[\mathcal{M}(D') \in O_i] + \delta_i,$$

where O_i are the outputs released to the ith analyst.

The multi-analyst DP variant supports the composition across different algorithms, as indicated by the following theorem.

Theorem 3.1 (Multi-DP Composition). *Given two randomized mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1 : \mathcal{D} \to (O_1, \ldots, O_m)$ satisfies $[(A_1, \epsilon_1, \delta_1), ..., (A_m, \epsilon_m, \delta_m)]$-multi-analyst-DP, and $\mathcal{M}_2 : \mathcal{D} \to (O'_1, \ldots, O'_m)$ satisfies $[(A_1, \epsilon'_1, \delta'_1), ..., (A_m, \epsilon'_m, \delta'_m)]$-multi-analyst-DP, then the mechanism $g(\mathcal{M}_1, \mathcal{M}_2)$ gives the $[(A_1, \epsilon_1 + \epsilon'_1, \delta_1 + \delta'_1), ..., (A_m, \epsilon_m + \epsilon'_m, \delta_m + \delta'_m)]$-multi-analyst-DP guarantee.*

Proof. As a sketch of proof, we note that by our definition of multi-analyst DP, if $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfies the notion of multi-analyst DP, then on each coordinate (i.e., for each data analyst), the mechanism provides DP guarantee according to each data analyst's privacy budget. Applying the sequential composition theorem (Theorem 2.1) to each coordinate, we can get this composition upper bound for multi-analyst DP. □

Under this new multi-analyst DP framework, several research questions (RQs) are raised and problem setups are considered.

RQ 1: worst-case privacy analysis across analysts. Under this multi-analyst DP framework, what if all data analysts collude or are compromised by an adversary, how could we design algorithms to account for the privacy loss to the colluded analysts? When this happens, we can obtain the following trivial lower bound and upper bound for the standard DP measure.

Theorem 3.2 (Compromisation Lower/Upper Bound, Trivial). *Given a mechanism $\mathcal{M}$ that satisfies $[(A_1, \epsilon_1, \delta_1), ..., (A_n, \epsilon_n, \delta_n)]$-multi-analyst-DP, when all data analysts collude, its DP loss is (i) lowered bounded by $(\max \epsilon_i, \max \delta_i)$, where (ϵ_i, δ_i) is the privacy loss to the i-th analyst, and (ii) trivially upper bounded by $(\sum \epsilon_i, \sum \delta_i)$.*

Obviously, this trivial upper bound does not match the lower bound, rendering the question of designing multi-analyst DP mechanisms to close the gap. In this paper, we would like to design such algorithms that achieve this lower bound, as shown in Section 5.

RQ 2: dynamic budget allocation across views. The DP query processing system should impose constraints on the total privacy loss by all the analysts (in the worst case) and the privacy loss per analyst. When handling incoming queries, prior work either dynamically allocates the privacy budget based on the budget request per query [29, 40] or query accuracy requirements [24] or predefine a set of static DP views that can handle the incoming queries [35].

The dynamic budget per query approach can deplete the privacy budget quickly as each query is handled with a fresh new privacy budget. The static views spend the budget in advance among them to handle unlimited queries, but they may fail to meet the accuracy requirements of some future queries. Therefore, in our work, we consider the view approach but assign budgets dynamically to the views based on the incoming queries so that more queries can be answered with their accuracy requirements. Specifically, we would like to use the histogram view, which queries the number of tuples in a database for each possible value of a set of attributes. The answer to a view is called a synopsis. We consider a set of views that can answer all incoming queries.

Definition 6 (Query Answerability [35]). *For a query q over the database D, if there exists a query q' over the histogram view V such that $q(D) = q(V(D))$, we say that q is answerable over V.*

Example 1. Consider two queries q_1 and q_2 over a database for employees Figure 1. They are answerable over the V_1, a 3-way marginal contingency table over attributes (age, gender, education), via their respective transformed queries $\hat{q}_1$ and $\hat{q}_2$. □

Given a set of views, we would like to design algorithms that can dynamically allocate privacy budgets to them and update their corresponding DP synopses over time. We show how these algorithms maximize the queries that can be handled accurately in Section 5. Since we can dynamically allocate budget to views, our system can add new views to the system as overtime. We discuss this possibility in Section 5.3.

RQ 3: fair query answering among data analysts. A fair system expects data analysts with higher privacy privileges to receive more accurate answers or larger privacy budgets than ones with lower privacy privileges. However, existing DP systems make no distinctions among data analysts. Hence, it is possible that a low-privilege external analyst who asks queries first consumes all the privacy budget and receives more accurate query answers, leaving no privacy budgets for high-privilege internal data analysts. It is also impossible to force data analysts to ask queries in a certain order. In this context, we would like the system to set up the (available and consumed) privacy budgets for data analysts according to their privacy privilege level. In particular, we define privacy privilege levels as an integer in the range of 1 to 10, where a higher number represents a higher privilege level. We also define a fairness notion inspired by the literature on resource allocation [3, 32, 49].

Definition 7 (Proportional Fairness). *Consider a DP system handling a sequence of queries Q from multiple data analysts with a mechanism $\mathcal{M}$, where each data analyst A_i is associated with a privilege level l_i. We say the mechanism $\mathcal{M}$ satisfies proportional fairness, if $\forall A_i, A_j$ $(i \neq j)$, $l_i \leq l_j$, we have*

$$\frac{Err_i(M, A_i, Q)}{\mu(l_i)} \leq \frac{Err_j(M, A_j, Q)}{\mu(l_j)},$$

where $Err_i(M, A_i, Q)$ denotes the analyst A_i's privacy budget consumption and $\mu(\cdot)$ is some linear function.

This fairness notion suggests the quality of query answers to data analysts, denoted by $Err_i(M, A_i, Q)$ is proportional to their privilege levels, denoted by a linear function $\mu(\cdot)$ of their privilege levels. We first consider the privacy budget per data analyst as the quality function, as a smaller error to the query answer is expected with a larger privacy budget. We show in Section 5.3 how to set up the system to achieve fairness when the analysts ask a sufficient number of queries, which means they finish consuming their assigned privacy budget.

4 SYSTEM OVERVIEW

In this section, we outline the key design principles of *DProvDB* and briefly describe the modules of the system.

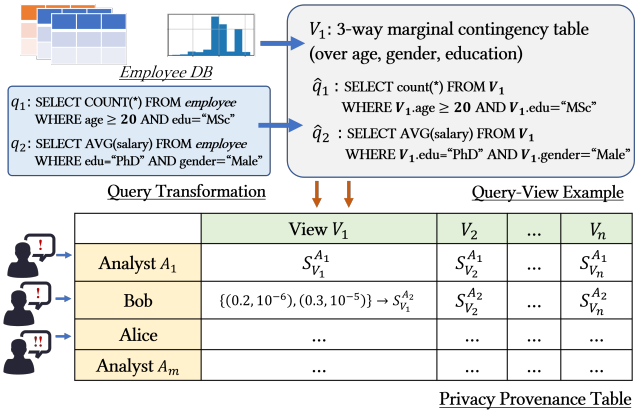

Figure 1: Illustration of the Histogram View, the Query Transformation and the Privacy Provenance Table: ① Histogram view V_1 over age, gender, and education is built on the database snapshot; ② Forthcoming queries q_1 and q_2 are transformed into linearly answerable queries $\hat{q}_1$ and $\hat{q}_2$ over V_1; ③ Analysts (A_1, A_2 with low, A_3 with high privilege) are recorded in the provenance table – the privacy loss for each view to each analyst is tracked for real-time queries.

4.1 Key Design Principles

To support the multi-analyst use case and to answer the aforementioned research questions, we identify the four principles and propose a system *DProvDB* that follows these principles.

Principle 1: fine-grained privacy provenance. The query processing system should be able to track the privacy budget allocated per each data analyst and per each view in a fine-grained way. The system should additionally enable a mechanism to compose privacy loss across data analysts and the queries they ask.

Principle 2: view-based privacy management. The queries are answered based on DP views or synopses. Compared to directly answering a query from the database D, view-based query answering can answer more private queries [35], but it assumes the accessibility of a pre-known query workload. In our system, the view is the minimum data object that we keep track of its privacy loss, and the views can be updated dynamically if higher data utility is required. The privacy budgets spent on different views during the updating process depend on the incoming queries.

Principle 3: dual query submission mode. Besides allowing data analysts to submit a budget with their query, the system enables an accuracy-aware mode. With this mode, data analysts can submit the query with their desired accuracy levels regarding the expected squared error. The dual mode system supports data analysts from domain experts, who can take full advantage of the budgets, to DP novices, who only care about the accuracy bounds of the query.

Principle 4: maximum query answering. The system should be tuned to answer as many queries accurately as possible without violating the privacy constraint specified by the administrator as per data analyst and per view based on their privilege levels.

4.2 Privacy Provenance Table

To meet the first two principles, we propose a privacy provenance table for *DProvDB*, inspired by the access matrix model in access control literature [15], to track the privacy loss per analyst and per view, and further bound the privacy loss. Particularly, in our model, the state of the overall privacy loss of the system is defined as a triplet $(\mathcal{A}, \mathcal{V}, \mathcal{P})$, where $\mathcal{A}$ denotes the set of data analysts and $\mathcal{V}$ represents the list of query-views maintained by the system. We denote by $\mathcal{P}$ the privacy provenance table, defined as follows.

Definition 8 (Privacy Provenance Table). *The privacy provenance table $\mathcal{P}$ consists of (i) a provenance matrix P that tracks the privacy loss of a view in $\mathcal{V}$ to each data analyst in $\mathcal{A}$, where each entry of the matrix $P[A_i, V_j]$ records the current cumulative privacy loss $S_{V_j}^{A_i}$, on view V_j to analyst A_i; (ii) a set of row/column/table constraints, Ψ: a* row constraint *for i-th row of P, denoted by ψ_{A_i}, refers to the allowed maximum privacy loss to a data analyst $A_i \in \mathcal{A}$ (according to his/her privilege level); a* column constraint *for the j-th column, denoted by ψ_{V_j} refers to as the allowed maximum privacy loss to a specific view V_j; the* table constraint *over P, denoted by ψ_P, specifies the overall privacy loss allowed for the protected database.*

The privacy constraints and the provenance matrix are correlated. In particular, the row/column constraints cannot exceed the overall table constraint, and each entry of the matrix cannot exceed row/column constraints. The correlations, such as the composition of the privacy constraints of all views or all analysts, depend on the DP mechanisms supported by the system. We provide the details of DP mechanisms and the respective correlations in privacy provenance table in Section 5.

Example 2. Figure 1 gives an example of the privacy provenance table for n views and m data analysts. When *DProvDB* receives query q_1 from Bob, it plans to use view V_1 to answer it. *DProvDB* first retrieves the previous cumulative cost of V_1 to Bob from the matrix, $P[Bob, V_1]$, and then computes the new cumulative cost $S_{V_1}^{Bob}$ for V_1 to Bob as if it answers q_1 using V_1. If the new cost $S_{V_1}^{Bob}$ is smaller than Bob's privacy constraint ψ_{Bob}, the view constraint ψ_{V_1}, and the table constraint ψ_P, *DProvDB* will answer q_1 and update $P[Bob, V_1]$ to $S_{V_1}^{Bob}$; otherwise, q_1 will be rejected. □

Due to the privacy constraints imposed by the privacy provenance table, queries can be rejected when the cumulative privacy cost exceeds the constraints. *DProvDB* needs to design DP mechanisms that well utilize the privacy budget to answer more queries. Hence, we formulate the *maximum query answering problem* based on the privacy provenance table.

Problem 1. *Given a privacy provenance table $(\mathcal{A}, \mathcal{V}, \mathcal{P})$, at each time, a data analyst $A_i \in \mathcal{A}$ submits the query with a utility requirement (q_i, v_i), where the transformed $\hat{q}_i \in \mathcal{V}$, how can we design a system to answer as many queries as possible without violating the row/column/table privacy constraints in P while meeting the utility requirement per query?*

4.3 System Modules

The *DProvDB* system works as a middle-ware between data analysts and existing DP DBMS systems (such as PINQ, Chorus, and

Algorithm 1: System Overview

Input: Analysts $\mathcal{A} := A_1, \ldots, A_n$; Database instance D; Privacy provenance table $\mathcal{P} = (P, \Psi)$.

1 Administrator sets up the privacy provenance table $\mathcal{P}$
2 Initialize all the synopses for $V \in \mathcal{V}$
3 **repeat**
4 Receive (q_i, v_i) from data analyst A_i
5 Select view $V \in \mathcal{V}$ to answer query q_i
6 Select mechanism $M \in \mathcal{M}$ applicable to q_i
7 $\epsilon_i \leftarrow$ M.privacyTranslate(q_i, v_i, V, p)
8 **if** M.constraintCheck$(P, A_i, V, \epsilon_i, \Psi)$ **then**
9 $V^{\epsilon'}_{A_i} \leftarrow$M.run$(P, A_i, V, \epsilon_i)$
10 Answer q_i with $V^{\epsilon'}_{A_i}$ and return answer r_i to A_i
11 **else**
12 reject the query q_i
13 **end**
14 **until** *No more queries sent by analysts*

PrivateSQL) to provide intriguing and add-on functionalities, including fine-grained privacy tracking, view/synopsis management, and privacy-accuracy translation. We briefly summarize the high-level ideas of the modules below.

Privacy Provenance Tracking. *DProvDB* maintains the privacy provenance table for each registered analyst and each generated view, as introduced in Section 4.2. Constraint checking is enabled based on this provenance tracking to decide whether to reject an analyst's query or not. We further build DP mechanisms to maintain and update the DP synopses and the privacy provenance table.

Dual Query Submission Mode. *DProvDB* provides two query submission modes to data analysts. *Privacy-oriented mode* [28, 29, 40, 59]: queries are submitted with a pre-apportioned privacy budget, i.e., $(A_i, q_i, \{\epsilon_i, \delta_i\})$. *Accuracy-oriented mode* [24, 51]: analysts can write the query with a desired accuracy bound, i.e., queries are in form of (A_i, q_i, v_i). We illustrate our algorithm with the accuracy-oriented mode.

Algorithm Overview. Algorithm 1 summarizes how *DProvDB* uses the DP synopses to answer incoming queries. At the system setup phase (line 1-2), the administrator initializes the privacy provenance table by setting the matrix entry as 0 and the row/column/table constraints Ψ. The system initializes empty synopses for each view. The data analyst specifies a query q_i with its desired utility requirement v_i (line 4). Once the system receives the request, it selects the suitable view and mechanism to answer this query (line 5-6) and uses the function privacyTranslate() to find the minimum privacy budget ϵ_i for V to meet the utility requirement of q_i (line 7). Then, *DProvDB* checks if answering q_i with budget ϵ_i will violate the privacy constraints Ψ (Line 8). If this constraint check passes, we run the mechanism to obtain a noisy synopsis (line 9). *DProvDB* uses this synopsis to answer query q_i and returns the answer to the data analyst (line 10). If the constraint check fails, *DProvDB* rejects the query (line 12). We show concrete DP mechanisms with their corresponding interfaces in the next section.

Algorithm 2: Vanilla Approach

1 Set δ in the system
2 **Function** run(P, A_i, V, ϵ_i) **:**
3 Generate a synopsis $V^{\epsilon_i}_{A_i}$ from view V
4 Update privacy provenance table $P[A_i, V] \leftarrow P[A_i, V] + \epsilon_i$
5 **return** $r_i \leftarrow V^{\epsilon_i}_{A_i}$
6 **end**
7 **Function** privacyTranslate(q_i, v_i, V, t) **:**
8 Set $u = \psi_P,\ l = 0$
9 $v \leftarrow$ calculateVariance(q_i, v_i, V)
10 $\epsilon =$ binarySearch$(l, u,$testAccuracy$(\cdot, v, \Delta q_i, \delta), t)$
11 **return** ϵ
12 **end**
13 **Function** constraintCheck$(P, A_i, V_j, \epsilon_i, \Psi)$ **:**
14 **if** $(P.\text{composite}() + \epsilon_i \le \Psi.\psi_P) \wedge$ $(P.\text{composite}(axis{=}Row) + \epsilon_i \le \Psi.\psi_{A_i}) \wedge$ $(P.\text{composite}(axis{=}Column) + \epsilon_i \le \Psi.\psi_{V_j})$ **then**
15 **return** True/Pass
16 **end**

5 DP ALGORITHM DESIGN

In this section, we first describe a vanilla DP mechanism that can instantiate the system interface but cannot maximize the number of queries being answered. Then we propose an additive Gaussian mechanism that leverages the correlated noise in query answering to improve the utility of the vanilla mechanism. Without loss of generality, we assume the data analysts do not submit the same query with decreased accuracy requirement (as they would be only interested in a more accurate query result).

5.1 Vanilla Approach

The vanilla approach is based on the Gaussian mechanism (applied to both the basic Gaussian mechanism [21] and the analytic Gaussian mechanism [2]). We describe how the system modules are instantiated with the vanilla approach.

5.1.1 Accuracy-Privacy Translation. This module interprets the user-specified utility requirement into the minimum privacy budget. For the vanilla mechanism, we use the following analytic Gaussian translation algorithm that takes in the query q_i and utility requirement v_i as input and outputs the minimum privacy budget to satisfy the utility requirement.

Definition 9 (Analytic Gaussian Translation). *Given a query $q : \mathcal{D} \to \mathbb{R}^d$ to achieve an expected squared error bound v for this query, the minimum privacy budget for the analytic Gaussian mechanism should satisfy $\Phi_{\mathcal{N}}\left(\frac{\Delta q}{2v} - \frac{\epsilon v}{\Delta q}\right) - e^{\epsilon}\Phi_{\mathcal{N}}\left(-\frac{\Delta q}{2v} - \frac{\epsilon v}{\Delta q}\right) \le \delta$. That is, given $\Delta q, \delta, v$, to solve the following optimization problem to find the minimal ϵ.*

$$\min_{\epsilon \in (0, \psi_P]} \epsilon \ s.t.\ \Phi_{\mathcal{N}}\left(\frac{\Delta q}{2v} - \frac{\epsilon v}{\Delta q}\right) - e^{\epsilon}\Phi_{\mathcal{N}}\left(-\frac{\Delta q}{2v} - \frac{\epsilon v}{\Delta q}\right) \le \delta \qquad (1)$$

Finding a closed-form solution to the problem above is not easy. However, we observe that the LHS of the constraint in Equation (1)

is a monotonic function of ϵ. Thus, we use binary search (Algorithm 2: 7-11) to search for the desired solution with a tolerance bound p on the accuracy, called the *translation precision*.

Additionally, we cannot directly use the accuracy bound the data analyst specified over the query as the error bound in Equation (1). This is because, instead of adding noise to the query result directly, we add noise to the view and then answer the query based on the noisy synopsis. Answering the query over the noisy synopsis may require adding up bins, which will scale up the noise variance. Therefore, we need to use the per query sensitivity and the view structure to calculate the error bound (line 9 in Algorithm 2) and then run the search algorithm.

5.1.2 Provenance Constraint Checking. As mentioned, the administrator can specify privacy constraints in privacy provenance table. *DProvDB* decide whether *reject or answer a query* using the provenance matrix P and the privacy constraints Ψ in privacy provenance table, as indicated in Algorithm 2: 13-15 (the function constraintCheck). This function checks whether the three types of constraints would be violated when the current query was to issue. The composite function in this constraint-checking algorithm can refer to the basic sequential composition or tighter privacy composition given by Renyi-DP [41] or zCDP [7, 22], see Appendix A for definitions. We suggest using the advanced composition for accounting privacy loss over time, but not for checking constraints, because the size of the provenance table $n * m$ is too small for a tight bound by this composition.

5.1.3 Putting Components All Together. The vanilla approach is aligned with existing DP query systems in the sense that it adds independent noise to the result of each query. Hence, it can be quickly integrated into these systems to provide privacy provenance and accuracy-aware features with little overhead. Algorithm 2: 2-5 (the function run) outlines how the vanilla method runs. It first generates the DP synopsis $V_{A_i}^{\epsilon_i}$ using analytic Gaussian mechanism for the chosen view V and updates the corresponding entry $P[A_i, V]$ in the privacy provenance table by compositing the consumed privacy loss (ϵ_i, δ_i) on the query (depending on the specific composition method used). We defer the analysis for the accuracy and privacy properties of the vanilla mechanism to Section 5.4.

5.2 Additive Gaussian Approach

We introduce a new additive Gaussian mechanism which is used for our synopses maintenance; then describe how *DProvDB* generates and updates the (local and global) DP synopses with this algorithm.

5.2.1 Additive Gaussian Mechanism. The additive Gaussian mechanism (additive GM or aGM) modifies the standard Gaussian mechanism, based on the nice statistical property of the Gaussian distribution — the sum of i.i.d. normal random variables is still normally distributed. We outline this primitive mechanism in Algorithm 3. This primitive takes a query q, a database instance D, a set of privacy budgets $\mathcal{B}$ corresponding to the set of data analysts $\mathcal{A}$ as input, and this primitive outputs a noisy query result to each data analyst A_i, which consumes the corresponding privacy budget (ϵ_i, δ). Its key idea is only to execute the query (to get the true answer on the database) once, and cumulatively inject noises to previous noisy answers, when multiple data analysts ask the same query. In particular, we sort the privacy budget set specified by the analysts. Starting from the largest budget, we add noise w.r.t the Gaussian variance σ_i^2 calculated from the query sensitivity Δq and this budget (ϵ_i, δ). For the rest of the budgets in the set, we calculate the Gaussian variance σ_j^2 in the same approach but add noise w.r.t $\sigma_j^2 - \sigma_i^2$ to the previous noisy answer. The algorithm then returns the noisy query answer to each data analyst. The privacy guarantee of this primitive is stated as follows.

Algorithm 3: Additive Gaussian Noise Calibration

Input: Analysts $\mathcal{A} := A_1, \ldots, A_n$; A query q; Database instance D; A set of privacy budgets $\mathcal{B} := (\epsilon_1, \delta), (\epsilon_2, \delta), \ldots, (\epsilon_n, \delta)$.
Output: A set of noisy answers $r_1, r_2, \ldots, r_n$.

Function additiveGM($\mathcal{A}, \mathcal{B}, q, D$) **:**
- $r \leftarrow$ queryExec(q, D) ▷ Obtain true query answer.
- $\Delta q \leftarrow$ sensCalc(q) ▷ Sensitivity calculation.
- $\mathcal{B}' \leftarrow$ sort$(\mathcal{B}, \epsilon_i)$ ▷ Sort $\mathcal{B}$ on the desc order of ϵ's.
- $(\epsilon_i, \delta) \leftarrow$ pop$(\mathcal{B}')$ ▷ Pop the 1st element.
- $\sigma_i \leftarrow$ analyticGM$(\epsilon_i, \delta, \Delta q)$ ▷ [2]
- $r_i \leftarrow r + \eta_i \sim \mathcal{N}(0, \sigma_i^2)$ ▷ Add Gaussian noise.
- **while** $\mathcal{B}' \neq \emptyset$ **do**
  - $(\epsilon_j, \delta) \leftarrow$ pop$(\mathcal{B}')$
  - $\sigma_j \leftarrow$ analyticGM$(\epsilon_j, \delta, \Delta q)$ ▷ [2]
  - $r_j \leftarrow r_i + \eta_j \sim \mathcal{N}(0, \sigma_j^2 - \sigma_i^2)$;
- **end**
- **return** $\mathcal{R} := \{r_i | i \in [n]\}$;

end

Theorem 5.1. *Given a database D, a set of privacy budgets $\mathcal{B} := (\epsilon_1, \delta), (\epsilon_2, \delta), \ldots, (\epsilon_n, \delta)$ and a query q, the additive Gaussian mechanism (Algorithm 4) that returns a set of noisy answers $r_1, r_2, \ldots, r_n$ to each data analyst A_i satisfies $[(A_1, \epsilon_1, \delta), ..., (A_n, \epsilon_n, \delta)]$-multi-analyst-DP and $(\max\{\epsilon_1, \ldots, \epsilon_n\}, \delta)$-DP.*

Proof Sketch. To each data analyst A_i, the DP mechanism is equivalent to the standard Gaussian mechanism with a proper variance that satisfies (ϵ_i, δ)-DP. Since the data is looked at once, the $(\max\{\epsilon_1, \ldots, \epsilon_n\}, \delta)$-DP is guaranteed by post-processing. □

5.2.2 Synopses Management. We introduce the concept of global and local DP synopses and then discuss the updating process in our additive GM. A DP synopsis (or synopsis for short) is a noisy answer to a (histogram) view over a database instance. We first use the privacy-oriented mode to explain the synopses management for clarity, and then elaborate the accuracy-oriented mode in the accuracy-privacy translation module (Section 5.2.3).

Global and Local DP Synopses. To solve the maximum query answering problem, for each view $V \in \mathcal{V}$, *DProvDB* maintains a *global DP synopsis* with a cost of (ϵ, δ), denoted by $V^{\epsilon,\delta}(D)$ or V^{ϵ}, where D is the database instance. For simplicity, we drop δ by considering the same value for all δ and D. For this veiw, *DProvDB* also maintains a *local DP synopsis* for each analyst $A_i \in \mathcal{A}$, denoted by $V_{A_i}^{\epsilon'}$, where the local synopsis is always generated from the global synopsis V^{ϵ} of the view V by adding more noise. Hence, we would

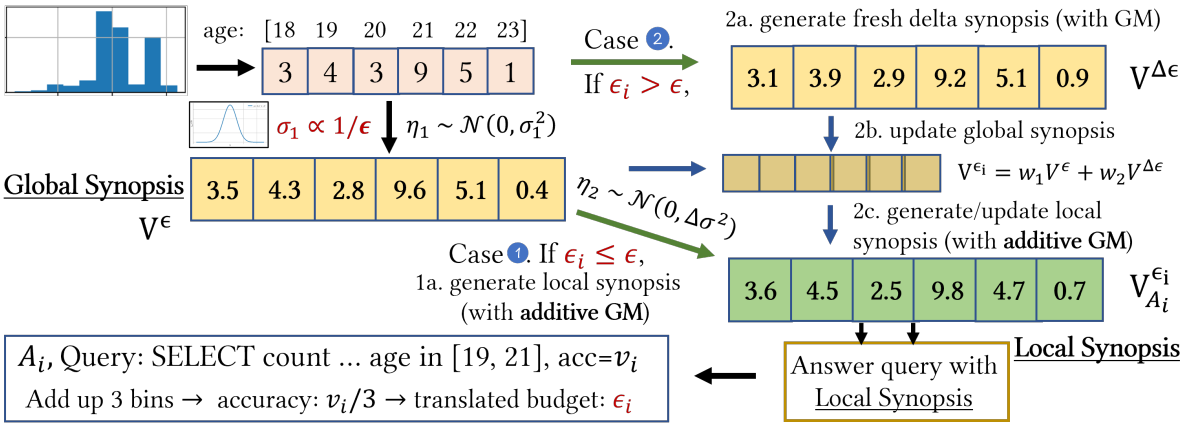

Figure 2: Illustration of the Additive Gaussian Approach

like to ensure $\epsilon \geq \epsilon'$. This local DP synopsis $V^{\epsilon'}_{A_i}$ will be used to answer the queries asked by the data analyst A_i.

The process of updating synopses consists of two parts. The first part is to update the local synopses based on the global synopses. The second part is to update the global synopses by relaxing the privacy guarantee, in order to answer a query with higher accuracy requirement. We discuss the details as below.

Generating Local Synopses from Global Synopses. We leverage our additive GM primitive to release a local DP synopsis $V^{\epsilon'}_{A_i}$ from a given global synopsis V^ϵ, where V^ϵ is generated by a Gaussian mechanism. Given the privacy guarantee ϵ (and δ) and the sensitivity of the view, the Gaussian mechanism can calculate a proper variance σ^2 for adding noise and ensuring DP. The additive GM calculates σ^2 and σ'^2 based on ϵ and ϵ' respectively, and then generates the local synopsis $V^{\epsilon'}_{A_i}$ by injecting independent noise drawn from $\mathcal{N}(0, \sigma'^2 - \sigma^2)$ to the global synopsis V^ϵ. As the global synopsis is hidden from all the analysts, the privacy loss to the analyst A_i is ϵ'. Even if all the analysts collude, the maximum privacy loss is bounded by the budget spent on the global synopsis.

Example 3. Alice and Bob are asking queries to *DProvDB*. Alice asks the first query q_1 (which is answerable on V_1) with *budget requirement* $\epsilon_{V_1,Alice} = 0.5$. *DProvDB* generates a global synopsis $V^{0.5}$ for V with budget 0.5 and then generate a local synopsis $V^{0.5}_{Alice}$ from the global synopsis $V^{0.5}$ for Alice. Bob next asks query q_2 (which is also answerable on V_1) with budget $\epsilon_{V_1,Bob} = 0.3$. Since the budget $0.3 < 0.5$, we use the additive GM to generate a local synopsis $V^{0.3}_{Bob}$ from the global synopsis $V^{0.5}$ for Bob and return the query answer based on the local synopsis $V^{0.3}_{Bob}$. This example follows "Case ①" in Fig. 2. □

Updating Global Synopses by Combining Views. When the global DP synopsis V^ϵ is not sufficiently accurate to handle a local synopsis request at privacy budget ϵ_t, *DProvDB* spends additional privacy budget $\Delta\epsilon$ to update the global DP synopsis to $V^{\epsilon+\Delta\epsilon}$, where $\Delta\epsilon = \epsilon_t - \epsilon$. We still consider Gaussian mechanism, which generates an intermediate DP synopsis $V^{\Delta\epsilon}$ with a budget $\Delta\epsilon$. Then we combine the previous synopses with this intermediate synopsis into an updated one. The key insight of the combination is to properly involve the fresh noisy synopses by assigning each synopsis with a weight proportional to the inverse of its noise variance, which gives the smallest expected square error based on UMVUE [33, 47]. That is, for the t-th release, we combine these two synopses:

$$V^{\epsilon_t} = (1 - w_t)V^{\epsilon_{t-1}} + w_t V^{\Delta\epsilon}. \tag{2}$$

The resulted expected square error for V^{ϵ_t} is $v_t = (1 - w_t)^2 v_{t-1} + w_t^2 v_\Delta$, where v_{t-1} is the noise variance of view $V^{\epsilon_{t-1}}$, and v_Δ is derived from $V^{\Delta\epsilon}$. To minimize the resulting error, $w_t = \frac{v_{t-1}}{v_\Delta + v_{t-1}}$.

Example 4. At the next time stamp, Bob asks a query q_1 with budget $\epsilon_{V_1,Bob} = 0.7$. Clearly the current global synopsis $V^{0.5}$ is not sufficient to answer this query because $0.7 > 0.5$. Then the system needs to update $V^{0.5}$ and this is done by: 1) first generating a fresh global synopsis $V^{0.2}$ using analytic GM from $V(D)$; 2) then combining it with $V^{0.5}$ to form $V^{0.7}$ using Equation (2). This example is illustrated as steps 2a and 2b of "Case ②" in Fig. 2. □

Lemma 5.2 (Correctness of View Combination). *Releasing the combined DP synopsis in the t-th update is $(\epsilon_{t-1} + \Delta\epsilon, \delta_{t-1} + \Delta\delta)$-DP.*

The view combination is not *frictionless*. Although the combined synopsis $V^{\epsilon+\Delta\epsilon}$ achieves $(\epsilon + \Delta\epsilon, \delta + \Delta\delta)$-DP, if we spend the whole privacy budget on generating a synopsis all at once, this one-time synopsis V^* has the same privacy guarantee but is with less expected error than $V^{\epsilon+\Delta\epsilon}$. We can show by sequentially combining and releasing synopses over time it is optimal among all possible linear combinations of synopses, however, designing a frictionless updating algorithm for Gaussian mechanisms is non-trivial in its own right, which remains for our future work.

Theorem 5.3 (Optimality of Linear View Combination). *Given a sequences of views, $V^{\epsilon_1}, V^{\Delta\epsilon_2}, \ldots, V^{\Delta\epsilon_t}$, the expected squared error of our t-th release is less than or equal to that of releasing $w'_1 V^{\epsilon_1} + \sum_{i=2}^{t} w'_i V^{\Delta\epsilon_i}$ for all $\{w_i \mid i = 1, \ldots, t\}$ where $\sum_i w'_i = 1$.*

Intuitively, this theorem is proved by a reduction to the best unbiased estimator and re-normalizing weights based on induction.

Updating Local Synopses and Accounting Privacy. When a local DP synopsis is not sufficiently accurate to handle a query, but the budget request for this query ϵ_i is still smaller than or

equal to the budget ϵ_t for the global synopsis of V, *DProvDB* generates a new local synopsis $V^{\epsilon_i}_{A_i}$ from V^{ϵ_t} using additive GM. The analyst A_i is able to combine query answers for a more accurate one, but the privacy cost for this analyst on view V is bounded as $\min(\epsilon_t, P[A_i, V] + \epsilon_i)$ which will be updated to $P[A_i, V]$.

Example 5. Analysts' queries are always answered on local synopses. To answer Bob's query $(q_1, \epsilon_{V_1,Bob} = 0.7)$, *DProvDB* uses additive GM to generate a fresh local synopsis $V^{0.7}_{Bob}$ from $V^{0.7}$ and return the answer to Bob. Alice asks another query $(q_1, \epsilon_{V_1,Alice} = 0.6)$. *DProvDB* updates $V^{0.5}_{Alice}$ by generating $V^{0.6}_{Alice}$ from $V^{0.7}$. Both analysts' privacy loss on V will be accounted as 0.7. This example complements the illustration of step 2c of "Case ②" in Fig. 2. □

5.2.3 Accuracy-Privacy Translation. The accuracy translation algorithm should consider the friction at the combination of global synopses. We propose an accuracy-privacy translation paradigm (Algorithm 4: 12) involving this consideration. This translator module takes the query q_i, the utility requirement v_i, the synopsis V for answering the query, and additionally the current global synopsis V^ϵ (we simplify the interface in Algorithm 1) as input, and outputs the corresponding privacy budget ϵ_i (omitting the same value δ).

As the first query release for the view does not involve the frictional updating issue, we separate the translation into two phases where the first query release directly follows the analytic Gaussian translation in our vanilla approach. For the second phase, given a global DP synopsis V^{ϵ_c} at hand (with *tracked* expected error v') for a specific query-view and a new query is submitted by a data analyst with expected error $v < v'$, we solve an optimization problem to find the Gaussian variance of the fresh new DP synopsis. We first calculate the Gaussian variance of the current DP synopsis v' (line 13) and solve the following problem (line 14-15).

$$\arg\max_{\sigma} \; v = w^2 v' + (1-w)^2 v_t \;\; \text{s.t. } w \in [0,1] \tag{3}$$

The solution gives us the minimal error variance $v_t = \sigma^2$ (line 15). By translating σ^2 into the privacy budget using the standard analytic Gaussian translation technique (Def. 9), we can get the minimum privacy budget that achieves the required accuracy guarantee (line 16). Note that when the requested accuracy $v > v'$, the solution to this optimization problem is $w = 0$, which automatically degrades to the vanilla translation.

Theorem 5.4. *Given a query q and the historically released DP synopsis V^ϵ for answering q, the additive Gaussian approach (Algorithm 4) returns the result with the expected squared error at most v and satisfies DP with a minimal cost of* privacyTranslate$(q, v, V, V^\epsilon).\epsilon$.

5.2.4 Provenance Constraint Checking. The provenance constraint checking for the additive Gaussian approach (line 18) is similar to the module for the vanilla approach. We would like to highlight 3 differences. 1) Due to the use of the additive Gaussian mechanism, the composition across analysts on the same view is bounded as tight as the max $P[A_i, V], \forall A_i \in \mathcal{A}$. Therefore, we check the view constraint by taking the max over the column retrieved by the index of view V_j. 2) To check table constraint, we composite a vector where each element is the maximum recorded privacy budget over every column/view. 3) The new cumulative cost is no longer ϵ_i, but $\epsilon' = \min(\epsilon, P[A_i, V] + \epsilon_i) - P[A_i, V]$.

Algorithm 4: Additive Gaussian Approach

1 Set δ in the system
2 **Function** run$(q, \mathcal{P}, A_i, V_{A_i}, \epsilon_i)$:
3 **if** $\epsilon_i > \epsilon$ *for global synopsis* V^ϵ **then**
4 $\Delta\epsilon = \epsilon_i - \epsilon$
5 $V^{\Delta\epsilon} \leftarrow V + \eta \sim \mathcal{N}(0, \text{analyticGM}(\Delta\epsilon, \delta, 1)^2 I)$
6 Update global synopsis to $V^{\epsilon \leftarrow \epsilon_i}$ with $V^{\Delta\epsilon}$
7 $\_, V^{\epsilon_i}_{A_i} \leftarrow \text{additiveGM}(\{A_i, A_{j, V^\epsilon = V^\epsilon_{A_j}}\}, \{\epsilon_i, \epsilon\}, q, D)$
8 Update local synopsis to $V^{\epsilon_i}_{A_i}$
9 Update $P[A_i, V] \leftarrow \min(\epsilon, P[A_i, V] + \epsilon_i)$
10 **return** $r_i \leftarrow V^{\epsilon_i}_{A_i}$
11 **end**
12 **Function** privacyTranslate(q, v, V, V^ϵ, t) :
13 $v' \leftarrow \text{estimateError}(q, v, V^\epsilon)$
14 $w \leftarrow \text{minimizer}(\text{func}=-\frac{(v-w^2v')}{(1-w)^2}, \text{bounds}=[0,1])$
15 $\epsilon \leftarrow \text{privacyTranslate}(q, \frac{(v-w^2v')}{(1-w)^2}, V, t)$ ▷ C.f. Def 9.
16 **return** ϵ
17 **end**
18 **Function** constraintCheck$(P, A_i, V^\epsilon, \epsilon_i, \Psi)$:
19 $\epsilon' = \min(\epsilon, P[A_i, V] + \epsilon_i) - P[A_i, V]$
20 **if** $(P.\text{max}(axis=Column,\ c=V_j) + \epsilon' \le \Psi.\psi_{V_j}) \wedge$
 $(P.\text{composite}(axis=Row,\ r=P.\text{max}()) + \epsilon' \le \Psi.\psi_P) \wedge$
 $(P.\text{composite}(axis=Row) + \epsilon' \le \Psi.\psi_{A_i})$ **then**
21 **return** True/Pass
22 **end**

Theorem 5.5. *Given the same sequence of queries Q and at least 2 data analysts in the system, the number of queries answered by the additive Gaussian approach is more than or equal to that answered by vanilla approach.*

5.2.5 Putting Components All Together. The additive Gaussian approach is presented in Algorithm 4: 2-10 (the function run). At each time stamp, the system receives the query q from the analyst A_i and selects the view that this query can be answerable. If the translated budget ϵ_i is greater than the budget allocated to the global synopsis of that view (line 3), we generate a delta synopsis (lines 4-5) and update the global synopsis (line 6). Otherwise, we can use additive GM to generate the local synopsis based on the (updated) global synopsis (lines 7-9). We update the provenance table with the consumed budget ϵ_i (line 10) and answer the query based on the local synopsis to the analyst (line 11).

5.2.6 Discussion on Combining Local Synopses. We may also update a local synopsis $V^{\epsilon'}_{A_i}$ (upon request ϵ_i) is first to generate an intermediate local synopsis $V^{\Delta\epsilon}_{A_i}$ from V^{ϵ_t} using additive GM, where $\Delta\epsilon = \epsilon_i - \epsilon'$ and then combines $V^{\Delta\epsilon}_{A_i}$ with the previous local synopsis in a similar way as it does for the global synopses, which leads to a new local synopsis $V^{\epsilon'+\Delta\epsilon}_{A_i}$.

Example 6. To answer Bob's query $(q_1, \epsilon_{V_1,Bob} = 0.7)$, we can use additive GM to generate a fresh local synopsis $V^{0.4}_{Bob}$ from $V^{0.7}$, and combine $V^{0.4}_{Bob}$ with the existing $V^{0.3}_{Bob}$ to get $V^{0.7}_{Bob}$. □

However, unlike combining global synopses, these local synopses share correlated noise. We must solve a different optimization problem to find an unbiased estimator with minimum error. For example, given the last combined global synopsis (and its weights) $V' = w_{t-1}V^{\epsilon_{t-1}} + w_t V^{\epsilon_t - \epsilon_{t-1}}$, if we know $V_A^{\epsilon_{t-1}}$ is a fresh local synopsis generated from $V^{\epsilon_{t-1}}$, we consider using the weights k_{t-1}, k_t for local synopsis combination:

$$\begin{aligned} V'_A &= k_{t-1}V_A^{\epsilon_{t-1}} + k_t V_A^{\Delta\epsilon} \\ &= k_{t-1}(V^{\epsilon_{t-1}} + \eta_A^{t-1}) + k_t(w_{t-1}V^{\epsilon_{t-1}} + w_t V^{\epsilon_t - \epsilon_{t-1}} + \eta_A^t) \\ &= (k_{t-1} + k_t w_{t-1})V^{\epsilon_{t-1}} + k_t w_t V^{\epsilon_t - \epsilon_{t-1}} + k_{t-1}\eta_A^{t-1} + k_t \eta_A^t, \end{aligned}$$

where η_A^{t-1} and η_A^t are the noise added to the local synopses in additive GM with variance σ_{t-1}^2 and σ_t^2. We can find the adjusted weights that minimize the expected error for V'_A is $v_{A,t} = (k_{t-1} + k_t w_{t-1})^2 v_{t-1} + k_t^2 w_t^2 v_\Delta + k_{t-1}^2 \sigma_{t-1}^2 + k_t^2 \sigma_t^2$ subject to $k_{t-1} + k_t w_{t-1} + k_t w_t = 1$, using an optimization solver. Allowing the optimal combination of local synopses tightens the cumulative privacy cost of A_i on V, i.e., $\epsilon_i < \min(\epsilon_t, P[A_i, V] + \epsilon_i)$. However, if the existing local synopsis $V_A^{\epsilon_{t-1}}$ is a combined synopsis from a previous time stamp, the correct variance calculation requires a nested analysis on *from where the local synopses are generated and with what weights the global/local synopses are combined.* This renders too many parameters for *DProvDB* to keep track of and puts additional challenges for accuracy translation since solving an optimization problem with many weights is hard, if not intractable. For practical reasons, *DProvDB* adopts the approach described in Algorithm 4.

5.3 Configuring Provenance Table

In existing systems, the administrator is only responsible for setting a single parameter specifying the overall privacy budget. *DProvDB*, however, requires the administrator to configure more parameters.

5.3.1 Setting Analyst Constraints for Proportional Fairness. We first discuss guidelines to establish per-analyst constraints in *DProvDB*, by proposing two specifications that achieve proportional fairness.

Definition 10 (Constraints for Vanilla Approach). *For the vanilla approach, we propose to specify each analyst's constraint by proportional indicated normalization. That is, for the table-level constraint ψ_P and analyst A_i with privilege l_i, $\psi_{A_i} = \frac{l_i}{\sum_{j\in[n]} l_j}\psi_P$.*

Note that this proposed specification is not optimal for the additive Gaussian approach, since the maximum utilized budget will then be constrained by $\max \psi_{A_i} < \psi_P$ when more than 1 analyst is using the system. Instead, we propose the following specification.

Definition 11 (Constraints for Additive Gaussian). *For the additive Gaussian approach, each analyst A_i's constraint can be set to as $\frac{l_i}{l_{max}}\psi_P$, where l_{max} denotes the maximum privilege in the system.*

Comparing Two Specifications. We compare the two analyst-constraint specifications (Def. 10 and 11) with experiments in Section 7.2. Besides their empirical performance, the vanilla constraint specification (Def. 10) requires all data analysts to be registered in the system before the provenance table is set up. The additive Gaussian approach (with specification in Def. 11), however, allows for the inclusion of new data analysts at a later time.

5.3.2 Setting View Constraints for Dynamic Budget Allocation. Existing privacy budget allocator [35] adopts a static splitting strategy such that the per view privacy budget is equal or proportional to their sensitivity. *DProvDB* subsumes theirs by, similarly, setting the view constraints to be equal or proportional to the sensitivity of each view, i.e., $\{\psi_{V_j} | V_j \in \mathcal{V}\} = \{\lambda_{V_j} \cdot \epsilon / \hat{\Delta}_{V_j}\}_{\forall V_j \in \mathcal{V}}$, where $\hat{\Delta}_{V_j}$ is the upper bound of the sensitivity of the view V_j. We therefore propose the following water-filling view constraint specification for a better view budget allocation.

Definition 12 (Water-filling View Constraint Setting). *The table constraint ψ_P has been set up as a constant (i.e., the overall privacy budget) in the system. The administrator simply set all view constraints the same as the table constraint, $\psi_{V_j} := \psi_P, \forall V_j \in \mathcal{V}$.*

With water-filling constraint specification, the provenance constraint checking will solely depend on the table constraint and the analyst constraints. The overall privacy budget is then dynamically allocated to the views based on analysts' queries. Compared to existing budget allocation methods on views, our water-filling specification based on the privacy provenance table reflects the actual accuracy demands on different views. Thus it results in avoiding the waste of privacy budget and providing better utility: i) *DProvDB* spends fewer budgets on unpopular views, whose consumed budget is less than $\lambda_{V_j} \cdot \epsilon / \hat{\Delta}_{V_j}$; ii) *DProvDB* can answer queries whose translated privacy budget $\epsilon > \lambda_{V_j} \cdot \epsilon / \hat{\Delta}_{V_j}$. In addition, the water-filling specification allows *DProvDB* to add views over time.

5.4 Privacy and Fairness Guarantees

Theorem 5.6 (System Privacy Guarantee). *Given the privacy provenance table and its constraint specifications, $\Psi = \{\psi_{A_i} | A_i \in \mathcal{A}\} \cup \{\psi_{V_j} | V_j \in \mathcal{V}\} \cup \{\psi_P\}$, both mechanisms for DProvDB ensure $[\ldots, (A_i, \psi_{A_i}, \delta), \ldots]$-multi-analyst-DP; they also ensure $\min(\psi_{V_j}, \psi_P)$-DP for view $V_j \in \mathcal{V}$ and overall ψ_P-DP if all the data analysts collude.*

With the provenance table, *DProvDB* can achieve proportional fairness when data analysts submit a sufficient number of queries, stated as in the following theorem.

Theorem 5.7 (System Fairness Guarantee). *Given the privacy provenance table P and the described approaches of setting analyst constraints, both mechanisms achieve proportional fairness, when the data analysts finish consuming their assigned privacy budget.*

Due to space constraints, proofs in this section are deferred to Appendix B.

6 DISCUSSION

In this section, we discuss a weaker threat model of the multi-analyst corruption assumption, with which additional utility gain is possible. We also discuss the fairness-utility trade-off for *DProvDB*.

6.1 Relaxation of Multi-analyst Threat Model

So far we have studied that all data analysts can collude. A more practical setting is, perhaps, considering a subset of data analysts that are compromised by the adversary. This setting is common to see in the multi-party computation research community [19] (*a.k.a* active security [5, 13]), where t out of n participating parties

are assumed to be corrupted. We propose the following notion of (t, n)-compromised multi-analyst DP.

Definition 13 ((t, n)-compromised Multi-analyst Setting). *We say a multi-analyst setting is (t, n)-compromised, if there exist n data analysts where at most t of them are **malicious** (meaning they can collude in submitting queries and sharing the answers).*

The (t, n)-compromised multi-analyst setting makes weaker assumptions on the attackers. Under this setting, the privacy loss is upper bounded by $(\sum_t \epsilon_i, \sum_t \delta_i)$, which is the summation over t largest privacy budgets. However, we cannot do better than the current *DProvDB* algorithms with this weaker setting under worst-case privacy assumption.

Theorem 6.1 (Hardness on Worst-case Privacy). *Given a mechanism $\mathcal{M}$ which is $[(A_1, \epsilon_1, \delta_1), \ldots, (A_n, \epsilon_n, \delta_n)]$-multi-analyst-DP, the worst-case privacy loss under (t, n)-compromised multi-analyst differential privacy is lower bounded by $(\max \epsilon_i, \max \delta_i)$, which is the same as under the all-compromisation setting.*

Proof Sketch. Under the worst-case assumption, the analyst with the largest privacy budget $(\max \epsilon_i, \max \delta_i)$ is sampled within the t compromised data analysts. Then it is natural to see that the lower bound of the privacy loss is $(\max \epsilon_i, \max \delta_i)$. □

At first glance, the relaxation of the (t, n)-compromisation does not provide us with better bounds. A second look, however, suggests the additional trust we put in the data analyst averages the privacy loss in compromisation cases. Therefore, with this relaxed privacy assumption, it is possible to design mechanisms to achieve better utility using policy-driven privacy.

Policies for multi-analysts. Blowfish privacy [26] specifies different levels of protection over sensitive information in the curated database. In the spirit of Blowfish, we can use policies to specify different levels of trust in data analysts using *DProvDB*. The privacy policy on analysts is defined as the following corruption graph.

Definition 14 ((t, n)-Analysts Corruption Graph). *Given n data analysts and assuming (t, n)-compromised setting, we say an undirected graph $G = (V, E)$ is a (t, n)-analysts corruption graph if,*

- *Each node in the vertex set $v_i \in V$ represents an analyst A_i;*
- *An edge is presented in the edge set $e(v_i, v_j) \in E$ if data analysts A_i and A_j can collude;*
- *Every connected component in G has less than t nodes.*

The corruption graph models the prior belief of the policy designer (or DB administrator) to the system users. Groups of analysts are believed to be not compromised if they are in the disjoint sets of the corruption graph. Based on this corruption graph, we can specify the analysts' constraints as assigning the overall privacy budget ψ_P to each connected component. This can be done by solving a weighted multi-dimensional knapsack problem, minimizing the overall wasting of the privacy budget and with constraints to the per-component budgets. The analysts' privileges can be written as a weighted vector inputting to the algorithm. We note that a better constraint assignment is possible by considering distances/hops among nodes in the graph [9, 10, 55].

Theorem 6.2. *There exist mechanisms with (t, n)-multi-analyst DP perform at least as well as with multi-analyst DP.*

Proof. Given a (t, n)-analysts corruption graph G, we show a naïve budget/constraint that makes (t, n)-multi-analyst DP degrade to multi-analyst DP. Ignoring the graph structure, we split the overall privacy budget ψ_P and assign a portion to each node proportional to the privilege weight on each node. □

Let k be the number of disjoint connected components in G. Then we have at most $\lceil k \rceil \cdot \psi_P$ privacy budget to assign to this graph. Clearly, the mechanisms with (t, n)-multi-analyst DP achieve $(\lceil k \rceil \cdot \psi_P)$-DP. When $n > t$, we have more than 1 connected component, meaning the overall privacy budget we could spend is more than that in the all-compromisation scenario.

6.2 Comparison with Strawman Solutions

One may argue for various alternative solutions to the multi-analyst query processing problem, as opposed to our proposed system. We justify these possible alternatives and compare them with *DProvDB*.

Strawman #1: Sharing Synthetic Data. Recall that *DProvDB* generates global and local synopses from which different levels of noise are added with additive GM, and answers analysts' queries based on the local synopses. We show our proposed algorithm is optimal in using privacy budgets when all data analysts collude. One possible alternative may be just releasing the global synopses (or generating synthetic data using all privacy budgets) to every data analyst, which also can achieve optimality in the all-compromisation setting. We note that this solution is $\min(\psi_P, (\max \epsilon_i, \max \delta_i))$-DP (same as the guarantee provided by *DProvDB*), however, it does not achieve the notion of *Multi-analyst DP* (all data analysts get the same output).

Strawman #2: Pre-calculating Seeded Caches. To avoid the cost of running the algorithm in an online manner, one may consider equally splitting the privacy budgets into small ones and using additive GM to pre-compute all the global synopses and local synopses. This solution saves all synopses and for future queries, it can directly answer by finding the appropriate synopses. If the space cost is too large, alternatively one may store only the seeds to generate all the synopses from which a synopsis can be quickly created.

This scheme arguably achieves the desired properties of *DProvDB*, if one ignores the storage overhead incurred by the pre-computed synopses or seeds. However, for an online query processing system, it is usually unpredictable what queries and accuracy requirements the data analysts would submit to the system. This solution focuses on doing most of the calculations offline, which may, first, lose the precision in translating accuracy to privacy, leading to a trade-off between precision and processing time regarding privacy translation. We will show in experiments that the translation only happens when the query does not hit the existing cache (which is rare), and the per-query translation processing time is negligible. Second, to pre-compute all the synopses, one needs to pre-allocate privacy budgets to the synopses. We will show as well using empirical results that this approach achieves less utility than *DProvDB*.

7 EXPERIMENTAL EVALUATION

In this section, we compare *DProvDB* with baseline systems and conduct an ablation investigation for a better understanding of different components in *DProvDB*. Our goal is to show that *DProvDB*

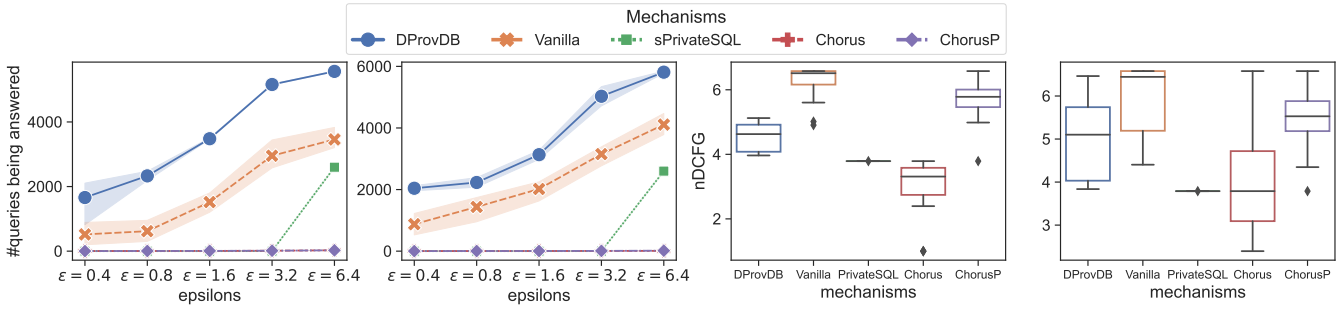

Figure 3: End-to-end Comparison (*RRQ* task, over *Adult* dataset), from left to right: a) utility v.s. overall budget, round-robin; b) utility v.s. overall budget, randomized; c) fairness against baselines, round-robin; d) fairness against baselines, randomized.

can improve existing query answering systems for multi-analyst in terms of the number of queries answered and fairness.

7.1 Experiment Setup

We implement *DProvDB* in Scala with PostgreSQL for the database system, and deploy it as a middle-ware between Chorus [28] and multi-analysts. Details can be found in Appendix C.

7.1.1 Baselines. Since we build on top of Chorus [28], we develop a number of baseline approaches for the multi-analysts framework using Chorus as well for fair comparisons.

- *Chorus [28]:* This baseline is the plain Chorus, which uses GM and makes no distinction between data analysts and uses no views. The system only sets the overall privacy budget.
- *DProvDB minus Cached Views (ChorusP):* This baseline enables privacy provenance tracking for each data analyst but does not store any synopses. We use Def. 10 to set up analyst constraints and Def. 12 for view constraints.
- *DProvDB minus Additive GM (Vanilla):* We equip Chorus with privacy provenance table and the cached views, but with our vanilla approach to update and manage the provenance table and the views. The privacy provenance table is configured the same as in ChorusP.
- *Simulating PrivateSQL (sPrivateSQL)[35]:* We simulate PrivateSQL by generating the static DP synopses at first. The budget allocated to each view is proportional to the view sensitivities [35]. Incoming queries that cannot be answered accurately with these synopses will be rejected.

7.1.2 Datasets and Use Cases. We use the Adult dataset [18] (a demographic data of 15 attributes and 45,224 rows) and the TPC-H dataset (a synthetic data of 1GB) [12] for experiments. We consider the following use cases.

- *Randomized range queries (RRQ):* We randomly generate 4,000 range queries *per analyst*, each with one attribute randomly selected with bias. Each query has the range specification $[s, s+o]$ where s and the offset o are drawn from a normal distribution. We design two types of query sequences from the data analysts: a) **round-robin**, where the analysts take turns to ask queries; b) **random**, where a data analyst is randomly selected each time.
- *Breadth-first search (BFS) tasks:* Each data analyst explores a dataset by traversing a decomposition tree of the cross product over the selected attributes, aiming to find the (sub-)regions with underrepresented records. That is, the data analyst traverses the domain and terminates only if the returned noisy count is within a specified threshold range.

To answer the queries, we generate one histogram view on each attribute. We use two analysts with privileges 1 and 4 by default setting and also study the effect of involving more analysts.

7.1.3 Evaluation Metrics. We use four metrics for evaluation.

- *Number of queries being answered*: We report the number of queries that can be answered by the system when no more queries can be answered as the utility metric to the system.
- *Cumulative privacy budget*: For BFS tasks that have fixed workloads, it is possible that the budget is not used up when the tasks are complete. Therefore, we report the total cumulative budget consumed by all data analysts when the tasks end.
- *Normalized discounted cumulative fairness gain (nDCFG)*: We coined an empirical fairness measure here. First, we introduce DCFG measure for a mechanism $\mathcal{M}$ as $\text{DCFG}_{\mathcal{M}} = \sum_{i=1}^{n} \frac{|Q_{A_i}|}{\log_2(\frac{1}{l_i}+1)}$, where l_i is the privilege level of analyst A_i and $|Q_{A_i}|$ is the total number of queries of A_i being answered. Then nDCFG is DCFG normalized by the total number of queries answered.
- *Runtime:* We measures the run time in milliseconds.

We repeat each experiment 4 times using different random seeds.

7.2 Empirical Results

We initiate experiments on the default settings of *DProvDB* and the baseline systems for comparison, and then study the effect of modifying the components of *DProvDB*.

7.2.1 End-to-end Comparison. This comparison is with the setup of the analyst constraints in line with Def. 11 for *DProvDB*, and with Def. 10 for vanilla approach. We brief our findings.

Results of RRQ task. We present Fig. 3 for this experiment. We fix the entire query workload and run the query processing on the five mechanisms. *DProvDB* outperforms all competing systems, in both round-robin or random use cases. Chorus and ChorusP can answer very few queries because their privacy budgets are depleted

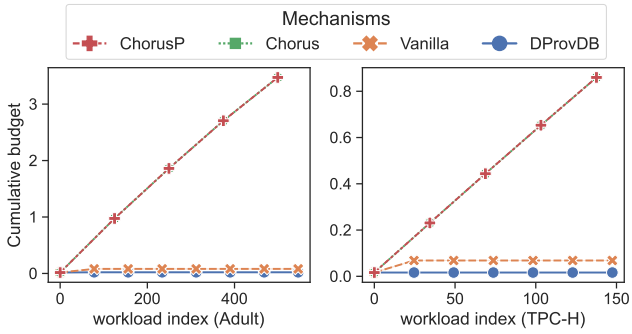

Figure 4: End-to-end Comparison (BFS task): Cumulative privacy budget assumption v.s. workload indices. Left: over *Adult* dataset; Right: over *TPC-H* dataset.

Table 2: Runtime Performance Comparison over Different Mechanisms on TPC-H Dataset (running on a Linux server with 64 GB DDR4 RAM and AMD Ryzen 5 3600 CPU, measured in milliseconds)

Systems	Setup Time	Running Time	No. of Queries	Per Query Perf
DProvDB	20388.65 ms	297.30 ms	86.0	3.46 ms
Vanilla	20388.65 ms	118.04 ms	86.0	1.37 ms
sPrivateSQL	20388.65 ms	166.51 ms	86.0	1.94 ms
Chorus	N/A	7380.47 ms	62.0	119.04 ms
ChorusP	N/A	7478.69 ms	62.0	120.62 ms

quickly. Interestingly, our simulated PrivateSQL can answer a number of queries which are comparable to *DProvDB* and vanilla when $\epsilon = 6.4$, but answers a limited number of queries under higher privacy regimes. This is intuitive because if one statically fairly pre-allocates the privacy budget to each view when the overall budget is limited, then possibly every synopsis can hardly answer queries accurately. One can also notice the fairness score of ChorusP is significantly higher than Chorus, meaning the enforcement of privacy provenance table can help achieve fairness.

Results of BFS task. An end-to-end comparison of the systems on the BFS tasks is depicted in Fig. 4. Both *DProvDB* and vanilla can complete executing the query workload with near constant budget consumption, while Chorus(P) spends privacy budget linearly with the workload size. *DProvDB* saves even more budget compared to vanilla approach, based on results on the TPC-H dataset.

Runtime performance. Table 2 summarizes the runtime of *DProvDB* and all baselines. *DProvDB* and sPrivateSQL use a rather large amount of time to set up views, however, answering large queries based on views saves more time on average compared to Chorus-based mechanisms. Recall that aGM requires solving optimization problems, where empirical results show incurred time overhead is only less than 2 ms per query.

We draw the same conclusion for our RRQ experiments and runtime performance test on the other dataset. We present the results in Appendix C.

7.2.2 Component Comparison. We consider three components in *DProvDB* to evaluate separately.

Cached Synopses. Given the same overall budget, mechanisms leveraging cached synopses outperform those without caches in terms of utility when the size of the query workload increases. This phenomenon can be observed for all budget settings, as shown in Fig. 5. Due to the use of cached synopses, *DProvDB* can potentially answer more queries if the incoming queries hit the caches. Thus, the number of queries being answered would increase with the increasing size of the workload, given the fixed overall budget.

Additive GM v.s. Vanilla. Given the same overall budget, additive GM outperforms the vanilla mechanism on utility. The utility gap increases with the increasing number of data analysts presented in the system. The empirical results are presented in Fig. 6. When there are 2 analysts in the system, additive GM can only gain a marginal advantage over vanilla approach; when the number of data analysts increases to 6, additive GM can answer around ~2x queries than vanilla. We also compare different settings of analyst constraints among different numbers of analysts and different overall system budgets (with 2 analysts). It turns out that additive GM with the setting in Def. 11 (DProvDB-l_max), is the best one, outperforming the setting 10 (DProvDB-l_sum, Vanilla-l_sum) all the time for different epsilons and with ~3x more queries being answered when #analysts=6.

Constraint Configuration. If we allow an expansion parameter $\tau \geq 1$ on setting the analyst constraints (i.e., overselling privacy budgets than the constraint defined in Def. 11), we can obtain higher system utility by sacrificing fairness, as shown in Fig. 7. With 2 data analysts, the number of total queries being answered by additive GM is slightly increased when we gradually set a larger analyst constraint expansion rate. Under a low privacy regime (viz., $\epsilon = 3.2$), this utility is increased by more than 15% comparing setting $\tau = 1.9$ and $\tau = 1.3$; on the other hand, the fairness score decreases by around 10%.

This result can be interpretable from a basic economic view, that we argue, as a system-wise public resource, the privacy budget could be *idle resources* when some of the data analysts stop asking queries. Thus, the *hard privacy constraints* enforced in the system can make some portion of privacy budgets unusable. Given the constraint expansion, we allow *DProvDB* to tweak between a fairness and utility trade-off, while the overall privacy is still guaranteed by the table constraint.

8 RELATED WORK

Enabling DP in query processing is an important line of research in database systems [42]. Starting from the first end-to-end interactive DP query system [40], existing work has been focused on generalizing to a larger class of database operators [1, 16, 17, 29, 44], building a programmable toolbox for experts [28, 59], or providing a user-friendly accuracy-aware interface for data exploration [24, 51]. Another line of research investigated means of saving privacy budgets in online query answering, including those based on cached views [35, 39] and the others based on injecting correlated noise to query results [4, 37, 58]. Privacy provenance (on protected data) is studied in the personalized DP framework [23] as a means to enforce varying protection on database records, which is dual to our system. The multi-analyst scenario is also studied in DP literature very recently, in terms of building a system to satisfy

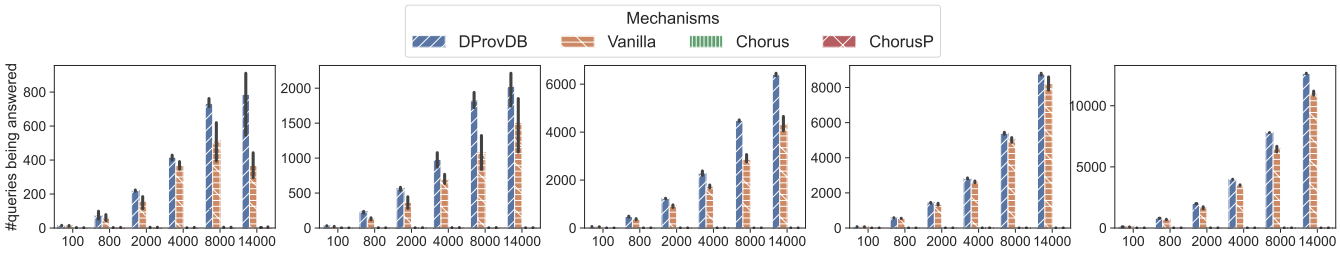

Figure 5: Component Comparison (*RRQ* task, over *Adult* dataset): Enabling Cached Synopses. Utility v.s. the size of query workload (round-robin)From left to right: $\epsilon = \{0.4, 0.8, 1.6, 3.2, 6.4\}$.

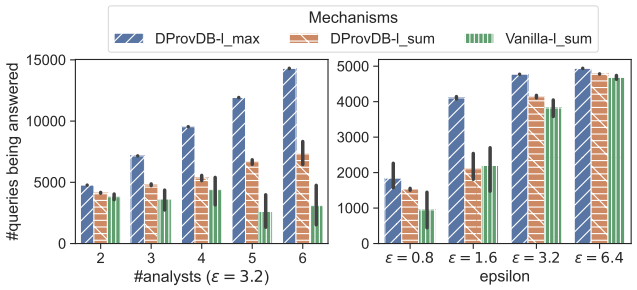

Figure 6: Component Comparison (*RRQ* task, over *Adult* dataset): Additive GM v.s. Vanilla. Left: utility v.s. #analysts, round-robin; Right: utility v.s. overall budgets, round-robin.

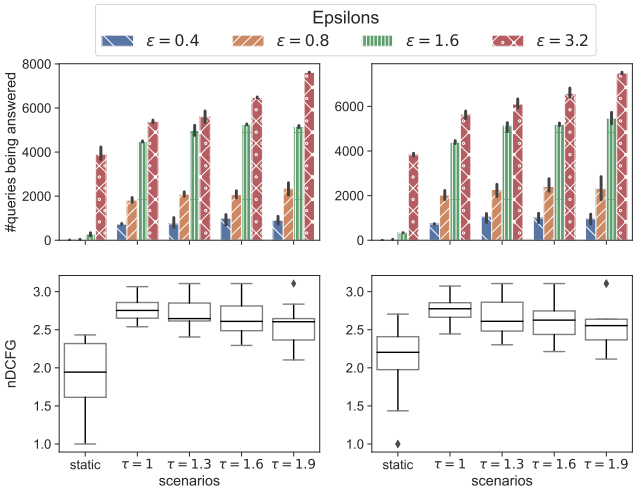

Figure 7: Component Comparison (*RRQ* task, over *Adult* dataset): Constraint Configuration. First row: utility v.s. constraint settings. Second row: fairness v.s. constraint settings. Left: round-robin. Right: randomized.

per-analyst privacy [57] or accuracy [34] constraints, or designing a mechanism to fairly answer queries among analysts [45, 46]. In their settings (either the offline [46] or online [45] version), fairness is defined as how well the (other) analysts' queries are answered or not interfered with when new data analyst is added to the system, which does not reflect the different privilege levels of analysts used in practice. None of the multi-analyst investigations, except this paper, consider the varying demands on queries from different data analysts and use this provenance information to maximize the number of queries answered and achieve proportional fairness.

9 CONCLUSION AND FUTURE WORK

We study how the query meta-data or provenance information can assist query processing in multi-analyst DP settings. We developed a privacy provenance framework for answering online queries, which tracks the privacy loss to each data analyst in the system using a provenance table. Based on the privacy provenance table, we proposed DP mechanisms that leverage the provenance information for noise calibration and built *DProvDB* to maximize the number of queries that can be answered. *DProvDB* can serve as a middle-ware to provide a multi-analyst interface for existing DP query answering systems. We implemented *DProvDB* and our evaluation shows that *DProvDB* can significantly improve the number of queries that could be answered and fairness, compared to baseline systems.

In the future, we would like to tighten the privacy analysis when composing the privacy loss of the local synopses generated from correlated global synopses. We can also optimize the system utility further by considering a more careful design of the structure of the cached synopses [39], e.g. making use of the sparsity in the data itself [56] or using data-dependent views [38]. Furthermore, this work opens the research on using provenance and query control for DP query processing. More research questions and works may be spawned by a deeper intertwinement between privacy provenance and access/leakage control [43], or focus on a more expressive model for privacy provenance, e.g., an analyst may be able to delegate his/her privacy privilege to other analysts temporarily.

REFERENCES

[1] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. 2022. Plume: Differential Privacy at Scale. *CoRR* abs/2201.11603 (2022). arXiv:2201.11603 https://arxiv.org/abs/2201.11603

[2] Borja Balle and Yu-Xiang Wang. 2018. Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 403–412. http://proceedings.mlr.press/v80/balle18a.html

[3] Siddhartha Banerjee, Vasilis Gkatzelis, Safwan Hossain, Billy Jin, Evi Micha, and Nisarg Shah. 2022. Proportionally Fair Online Allocation of Public Goods with Predictions. *CoRR* abs/2209.15305 (2022). https://doi.org/10.48550/arXiv.2209.15305 arXiv:2209.15305

[4] Ergute Bao, Yizheng Zhu, Xiaokui Xiao, Yin Yang, Beng Chin Ooi, Benjamin Hong Meng Tan, and Khin Mi Mi Aung. 2022. Skellam Mixture Mechanism: a Novel Approach to Federated Learning with Differential Privacy. *Proc. VLDB Endow.* 15, 11 (2022), 2348–2360. https://www.vldb.org/pvldb/vol15/p2348-bao.pdf

[5] Donald Beaver and Shafi Goldwasser. 1989. Multiparty Computation with Faulty Majority. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings (Lecture Notes in Computer Science, Vol. 435)*, Gilles Brassard (Ed.). Springer, 589–590. https://doi.org/10.1007/0-387-34805-0_51

[6] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2013. Differentially private data analysis of social networks via restricted sensitivity. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, Robert D. Kleinberg (Ed.). ACM, 87–96. https://doi.org/10.1145/2422436.2422449

[7] Mark Bun and Thomas Steinke. 2016. Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9985)*, Martin Hirt and Adam D. Smith (Eds.). 635–658. https://doi.org/10.1007/978-3-662-53641-4_24

[8] Peter Buneman and Wang Chiew Tan. 2007. Provenance in databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou (Eds.). ACM, 1171–1173. https://doi.org/10.1145/1247480.1247646

[9] Yang Cao, Shun Takagi, Yonghui Xiao, Li Xiong, and Masatoshi Yoshikawa. 2020. PANDA: Policy-aware Location Privacy for Epidemic Surveillance. *Proc. VLDB Endow.* 13, 12 (2020), 3001–3004. https://doi.org/10.14778/3415478.3415529

[10] Konstantinos Chatzikokolakis, Miguel E. Andrés, Nicolás Emilio Bordenabe, and Catuscia Palamidessi. 2013. Broadening the Scope of Differential Privacy Using Metrics. In *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7981)*, Emiliano De Cristofaro and Matthew K. Wright (Eds.). Springer, 82–102. https://doi.org/10.1007/978-3-642-39077-7_5

[11] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (2009), 379–474. https://doi.org/10.1561/1900000006

[12] The Transaction Processing Performance Council. 2008. The TPC Benchmark H (TPC-H). https://www.tpc.org/tpch/

[13] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4622)*, Alfred Menezes (Ed.). Springer, 572–590. https://doi.org/10.1007/978-3-540-74143-5_32

[14] Damien Desfontaines and Balázs Pejó. 2020. SoK: Differential privacies. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 288–313. https://doi.org/10.2478/popets-2020-0028

[15] Sabrina De Capitani di Vimercati. 2011. Access Control Policies, Models, and Mechanisms. In *Encyclopedia of Cryptography and Security, 2nd Ed*, Henk C. A. van Tilborg and Sushil Jajodia (Eds.). Springer, 13–14. https://doi.org/10.1007/978-1-4419-5906-5_806

[16] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 759–772. https://doi.org/10.1145/3514221.3517844

[17] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 432–444. https://doi.org/10.1145/3448016.3452813

[18] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[19] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4004)*, Serge Vaudenay (Ed.). Springer, 486–503. https://doi.org/10.1007/11761679_29

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, 265–284. https://doi.org/10.1007/11681878_14

[21] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407. https://doi.org/10.1561/0400000042

[22] Cynthia Dwork and Guy N. Rothblum. 2016. Concentrated Differential Privacy. *CoRR* abs/1603.01887 (2016). arXiv:1603.01887 http://arxiv.org/abs/1603.01887

[23] Hamid Ebadi, David Sands, and Gerardo Schneider. 2015. Differential Privacy: Now it's Getting Personal. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 69–81. https://doi.org/10.1145/2676726.2677005

[24] Chang Ge, Xi He, Ihab F. Ilyas, and Ashwin Machanavajjhala. 2019. APEx: Accuracy-Aware Differentially Private Data Exploration. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 177–194. https://doi.org/10.1145/3299869.3300092

[25] David Hall et al. 2021. Breeze: numerical processing library for Scala. https://github.com/scalanlp/breeze. Accessed: March 12, 2023.

[26] Xi He, Ashwin Machanavajjhala, and Bolin Ding. 2014. Blowfish privacy: tuning privacy-utility trade-offs using policies. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 1447–1458. https://doi.org/10.1145/2588555.2588581

[27] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446. https://doi.org/10.1145/582415.582418

[28] Noah M. Johnson, Joseph P. Near, Joseph M. Hellerstein, and Dawn Song. 2020. Chorus: a Programming Framework for Building Scalable Differential Privacy Mechanisms. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 535–551. https://doi.org/10.1109/EuroSP48549.2020.00041

[29] Noah M. Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. *Proc. VLDB Endow.* 11, 5 (2018), 526–539. https://doi.org/10.1145/3187009.3177733

[30] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2015. The Composition Theorem for Differential Privacy. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1376–1385. https://proceedings.mlr.press/v37/kairouz15.html

[31] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2017. The Composition Theorem for Differential Privacy. *IEEE Trans. Inf. Theory* 63, 6 (2017), 4037–4049. https://doi.org/10.1109/TIT.2017.2685505

[32] Frank P Kelly, Aman K Maulloo, and David Kim Hong Tan. 1998. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society* 49, 3 (1998), 237–252.

[33] J Kiefer. 1952. On minimum variance estimators. *The Annals of Mathematical Statistics* 23, 4 (1952), 627–629.

[34] Karl Knopf. 2021. Framework for Differentially Private Data Analysis with Multiple Accuracy Requirements. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2890–2892. https://doi.org/10.1145/3448016.3450587

[35] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. *Proc. VLDB Endow.* 12, 11 (2019), 1371–1384. https://doi.org/10.14778/3342263.3342274

[36] Ios Kotsogiannis, Yuchao Tao, Ashwin Machanavajjhala, Gerome Miklau, and Michael Hay. 2019. Architecting a Differentially Private SQL Engine. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p125-kotsogiannis-cidr19.pdf

[37] Fragkiskos Koufogiannis, Shuo Han, and George J. Pappas. 2016. Gradual Release of Sensitive Data under Differential Privacy. *J. Priv. Confidentiality* 7, 2 (2016). https://doi.org/10.29012/jpc.v7i2.649

[38] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. 2014. A Data- and Workload-Aware Query Answering Algorithm for Range Queries Under Differential Privacy. *Proc. VLDB Endow.* 7, 5 (2014), 341–352. https://doi.org/10.14778/2732269.2732271

[39] Miti Mazmudar, Thomas Humphries, Jiaxiang Liu, Matthew Rafuse, and Xi He. 2022. Cache Me If You Can: Accuracy-Aware Inference Engine for Differentially Private Data Exploration. *Proc. VLDB Endow.* 16, 4 (2022), 574–586. https://www.vldb.org/pvldb/vol16/p574-mazmudar.pdf

[40] Frank McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul (Eds.). ACM, 19–30. https://doi.org/10.1145/1559845.1559850

[41] Ilya Mironov. 2017. Rényi Differential Privacy. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 263–275. https://doi.org/10.1109/CSF.2017.11

[42] Joseph P. Near and Xi He. 2021. Differential Privacy for Databases. *Found. Trends Databases* 11, 2 (2021), 109–225. https://doi.org/10.1561/1900000066

[43] Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. 2022. Don't Be a Tattle-Tale: Preventing Leakages through Data Dependencies on Access Control Protected Data. *Proc. VLDB Endow.* 15, 11 (2022), 2437–2449. https://www.vldb.org/pvldb/vol15/p2437-pappachan.pdf

[44] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating Data to Sensitivity in Private Data Analysis. *Proc. VLDB Endow.* 7, 8 (2014), 637–648. https://doi.org/10.14778/2732296.2732300

[45] David Pujol, Albert Sun, Brandon Fain, and Ashwin Machanavajjhala. 2022. Multi-Analyst Differential Privacy for Online Query Answering. *Proc. VLDB Endow.* 16, 4 (2022), 816–828. https://www.vldb.org/pvldb/vol16/p816-pujol.pdf

[46] David Pujol, Yikai Wu, Brandon Fain, and Ashwin Machanavajjhala. 2021. Budget Sharing for Multi-Analyst Differential Privacy. *Proc. VLDB Endow.* 14, 10 (2021), 1805–1817. https://doi.org/10.14778/3467861.3467870

[47] C Radhakrishna Rao. 1949. Sufficient statistics and minimum variance estimates. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 45. Cambridge University Press, 213–218.

[48] Sofya Raskhodnikova and Adam D. Smith. 2015. Efficient Lipschitz Extensions for High-Dimensional Graph Statistics and Node Private Degree Distributions. *CoRR* abs/1504.07912 (2015). arXiv:1504.07912 http://arxiv.org/abs/1504.07912

[49] Julia Stoyanovich, Ke Yang, and H. V. Jagadish. 2018. Online Set Selection with Fairness and Diversity Constraints. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose (Eds.). OpenProceedings.org, 241–252. https://doi.org/10.5441/002/edbt.2018.22

[50] Tim van Erven and Peter Harremoës. 2014. Rényi Divergence and Kullback-Leibler Divergence. *IEEE Trans. Inf. Theory* 60, 7 (2014), 3797–3820. https://doi.org/10.1109/TIT.2014.2320500

[51] Elisabet Lobo Vesga, Alejandro Russo, and Marco Gaboardi. 2020. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 411–428. https://doi.org/10.1109/SP40000.2020.00086

[52] Paul Voigt and Axel Von dem Bussche. 2017. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10 (2017), 3152676.

[53] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Type Ranking Measures. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA (JMLR Workshop and Conference Proceedings, Vol. 30)*, Shai Shalev-Shwartz and Ingo Steinwart (Eds.). JMLR.org, 25–54. http://proceedings.mlr.press/v30/Wang13.html

[54] Royce J. Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2020. Differentially Private SQL with Bounded User Contribution. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 230–250. https://doi.org/10.2478/popets-2020-0025

[55] Zhuolun Xiang, Bolin Ding, Xi He, and Jingren Zhou. 2020. Linear and Range Counting under Metric-based Local Differential Privacy. In *IEEE International Symposium on Information Theory, ISIT 2020, Los Angeles, CA, USA, June 21-26, 2020*. IEEE, 908–913. https://doi.org/10.1109/ISIT44484.2020.9173952

[56] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2011. Differential Privacy via Wavelet Transforms. *IEEE Trans. Knowl. Data Eng.* 23, 8 (2011), 1200–1214. https://doi.org/10.1109/TKDE.2010.247

[57] Yingtai Xiao, Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Daniel Kifer. 2021. Optimizing Fitness-For-Use of Differentially Private Linear Queries. *Proc. VLDB Endow.* 14, 10 (2021), 1730–1742. https://doi.org/10.14778/3467861.3467864

[58] Yingtai Xiao, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2022. Answering Private Linear Queries Adaptively using the Common Mechanism. *CoRR* abs/2212.00135 (2022). https://doi.org/10.48550/arXiv.2212.00135 arXiv:2212.00135

[59] Dan Zhang, Ryan McKenna, Ios Kotsogiannis, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. 2018. EKTELO: A Framework for Defining Differentially-Private Computations. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 115–130. https://doi.org/10.1145/3183713.3196921

A OTHER DP NOTIONS

Other than the basic sequential composition, *DProvDB* also incorporates the advanced composition and Rényi DP composition. We provide the necessary definitions for these notions.

Theorem A.1 (Advanced Composition [30, 31]). *For any $\epsilon \geq 0$ and $\delta \in (0,1)$, the k-fold composition of (ϵ, δ)-DP mechanisms satisfies $((k-2i)\epsilon, 1-(1-\delta)^k(1-\delta_i))$-DP, for all $i = \{0, 1, \ldots, \lfloor k/2 \rfloor\}$, where $\delta_i = \frac{\sum_{\ell=0}^{i-1} \binom{k}{\ell}(e^{(k-\ell)\epsilon} - e^{(k-2i+\ell)\epsilon})}{(1+e^{\epsilon})^k}$.*

Definition 15 ((α, ϵ)-Rényi Differential Privacy [41]). *For $\alpha \geq 1$, we say a privacy mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{O}$ satisfies (α, ϵ)-Rényi-differential-privacy (or RDP) if for all pairs of neighbouring databases D and D' and all $O \subseteq \mathcal{O}$, we have*

$$\mathbb{E}_{O \sim \mathcal{M}(D)}[e^{(\alpha-1)\mathcal{L}_{D/D'}(O)}] \leq e^{(\alpha-1)\epsilon}.$$

When $\alpha \to 1$, $(1, \epsilon)$-RDP is defined to be equivalent to ϵ-KL privacy [14, 50].

Theorem A.2 (Rényi DP Composition [41]). *Given two mechanisms $\mathcal{M}_1 : \mathcal{D} \to \mathcal{O}_1$ and $\mathcal{M}_2 : \mathcal{D} \to \mathcal{O}_2$, s.t. $\mathcal{M}_1$ satisfies (α, ϵ_1)-RDP and $\mathcal{M}_2$ satisfies (α, ϵ_2)-RDP. The combination of the two mechanisms $\mathcal{M}_{1,2} : \mathcal{D} \to \mathcal{O}_1 \times \mathcal{O}_2$, which is a mapping $\mathcal{M}_{1,2}(D) = (\mathcal{M}_1(D), \mathcal{M}_2(D))$, is $(\alpha, \epsilon_1 + \epsilon_2)$-RDP.*

Theorem A.3 (Rényi DP to (ϵ, δ)-DP [41]). *If a mechanism $\mathcal{M}$ satisfies (α, ϵ)-RDP, it also satisfies $(\epsilon + \frac{\log 1/\delta}{\alpha - 1}, \delta)$-DP, for any $\delta \in (0, 1)$.*

B OMITTED PROOFS

B.1 Proof of Theorem 3.1

Proof of Theorem 3.1. As a sketch of proof, we note that by our definition of multi-analyst DP, if $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfies the notion of multi-analyst DP, then on each coordinate (i.e., for each data analyst), the mechanism provides DP guarantee according to each data analyst's privacy budget. Applying the sequential composition theorem (Theorem 2.1) to each coordinate, we can get this composition upper bound for multi-analyst DP. □

B.2 Proof of Theorem 5.1

Proof of Theorem 5.1. For each data analyst A_j, where $j \in [n]$, its corresponding privacy budget is (ϵ_j, δ) and the additive Gaussian mechanism $\mathcal{M}_{aGM}$ returns r_j to A_j. We first prove the additive Gaussian mechanism $\mathcal{M}_{aGM}$ satisfies multi-analyst-DP, that is, we need to show that,

$$\Pr[\mathcal{M}_{aGM}(D) = r_j] \leq e^{\epsilon_j} \Pr[\mathcal{M}_{aGM}(D') = r_j] + \delta_j. \quad (4)$$

Case 1. If $\epsilon_j = \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$, it is not hard to see the noise generation and addition to the query answer are the same as in the analytic Gaussian mechanism, and therefore equation 4 holds.

Case 2. If $\epsilon_j \neq \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$, w.l.o.g., we can assume $\epsilon_i = \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$. If we use analytic Gaussian mechanism to calculate the Gaussian variance, we have,

$$\Phi_{\mathcal{N}}\left(\frac{\Delta q}{2\sigma_i} - \frac{\epsilon_i \sigma_i}{\Delta q}\right) - e^{\epsilon_i}\Phi_{\mathcal{N}}\left(-\frac{\Delta q}{2\sigma_i} - \frac{\epsilon_i \sigma_i}{\Delta q}\right) \leq \delta$$

and

$$\Phi_{\mathcal{N}}\left(\frac{\Delta q}{2\sigma_j} - \frac{\epsilon_j \sigma_j}{\Delta q}\right) - e^{\epsilon_j}\Phi_{\mathcal{N}}\left(-\frac{\Delta q}{2\sigma_j} - \frac{\epsilon_j \sigma_j}{\Delta q}\right) \leq \delta.$$

Then for the noisy answer returned to data analyst A_j, we have $r_j := r_i + \eta_j = r + \eta_i + \eta_j$, where $\eta_i \sim \mathcal{N}(0, \sigma_i^2)$ and $\eta_j \sim \mathcal{N}(0, \sigma_j^2 - \sigma_i^2)$. Since η_i and η_j are independent random variables drawn from Gaussian distribution, we obtain $\eta_i + \eta_j \sim \mathcal{N}(0, \sigma_j^2)$. Thus, Equation (4) holds according to the analytic Gaussian mechanism (Def. 3).

Then we need to prove $\mathcal{M}_{aGM}$ satisfies $(\max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \delta)$-DP. In the additive Gaussian mechanism, we look at the true query answer (and the data) only once and add noise drawn from $\mathcal{N}(0, \sigma_i^2)$ to it (assuming $\epsilon_i = \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$). According to the post-processing theorem of DP (Theorem 2.2), the overall privacy guarantee is bounded by $(\max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \delta)$-DP. □

B.3 Proof of Theorem 5.6

Proof of Theorem 5.6. We analyze the privacy guarantee for the vanilla approach and the additive Gaussian approach separately.
Analysis of the vanilla approach. In vanilla approach, the accuracy requirement is translated to the minimal privacy cost (as in Def. 9) and recorded in the privacy provenance table P by composition tools if the sanity checking passes. The $[\ldots, (A_i, \psi_{A_i}, \delta), \ldots]$-multi-analyst-DP is guaranteed by the analyst constraint checking on P. The privacy loss on each view is either bounded by its own constraint ψ_{V_j} or bounded by the table constraint ψ_P for water-filling specification (Def. 12), so *DProvDB* guarantees $\min(\psi_{V_j}, \psi_P)$-DP for view V_j. The table constraint ψ_P bounds the overall information disclosure on the private database. Hence, even when all data analysts $A_1, \ldots, A_m$ collude, the vanilla mechanism is ψ_P-DP.
Analysis of the additive Gaussian approach. Recall that in the additive Gaussian approach, the noise added to different analysts' queries that can be answered on the same view is calibrated using additive GM. By Theorem 5.1, for each view V_j, the mechanism achieves $[\ldots, (A_i, \min(\psi_{A_i}, \psi_{V_j}), \delta), \ldots]$-multi-analyst-DP and ψ_{V_j}-DP for collusion. Composing for all views, the mechanism provides $[\ldots, (A_i, \psi_{A_i}, \delta), \ldots]$-multi-analyst-DP and provides $\min(\max \psi_{A_i}, \sum \psi_{V_j}) = \psi_P$-DP guarantee for the protected DB. Note that additive Gaussian approach use the same view constraint specification as in vanilla approach, therefore, provides the same per view privacy guarantee. □

B.4 Proof of Theorem 5.7

Proof of Theorem 5.7. Recall the proportional fairness says that for system or mechanism $\mathcal{M}$, $\forall A_i, A_j$ $(i \neq j)$, $l_i \leq l_j$, we have $\frac{Err_i(M, A_i, Q)}{\mu(l_i)} \leq \frac{Err_j(M, A_j, Q)}{\mu(l_j)}$. Note that $Err_i(M, A_i, Q)$ represents the amount of privacy budgets spent by analyst A_i. When a sufficient number of queries are submitted and answered, we have $\lim_{|Q|\to\infty} Err_i(M, A_i, Q) = \psi_{A_i}$. Then we show that the inequality holds for both analyst constraint specifications.

Analysis of the vanilla approach. (Def. 10) In vanilla approach, the analyst constraint of A_i is set as $\psi_{A_i} = \frac{l_i}{\sum_{j\in[n]} l_j}\psi_P$. For a given set of analysts and their privilege levels, $\frac{\psi_P}{\sum_{j\in[n]} l_j}$ is a constant, and therefore ψ_{A_i} is a linear function of l_i. Then for $l_i \leq l_j$, $\frac{Err_i(M,A_i,Q)}{\mu(l_i)} \leq \frac{Err_j(M,A_j,Q)}{\mu(l_j)}$. holds for any linear $\mu(\cdot)$.
Analysis of the additive Gaussian approach. (Def. 11) In the additive Gaussian approach, the analyst constraint of A_i is set as $\psi_{A_i} = \frac{l_i}{l_{max}}\psi_P$. Note that l_{max} is the maximum privilege in the system, not the maximum privilege of a set of analysts. Thus $\frac{\psi_P}{l_{max}}$ is also a constant which makes the inequality hold as well. □

B.5 Proof of Theorem 6.2

Proof of Theorem 6.2. Given a (t, n)-analysts corruption graph G, we show a naïve budget/constraint that makes (t, n)-multi-analyst DP degrade to multi-analyst DP. Ignoring the graph structure, we split the overall privacy budget ψ_P and assign a portion to each node proportional to the privilege weight on each node. □

C OTHER EXPERIMENTAL EVALUATION

C.1 Implementation Details

We implement *DProvDB* in Scala 2.12.2 and use sbt as the system dependency management tool. First, we re-implement the analytic Gaussian mechanism [2] library in Scala. Second, we use PostgreSQL as the underlying database platform and utilize the Breeze library[2] for noise calibration and solving the optimization problem in privacy translation (c.f. additive Gaussian approach). *DProvDB* works as a middleware between the data analysts and the existing DP SQL query processing system, and provides advanced functionalities such as privacy provenance, query control, query answering over the cached views, and privacy translation. *DProvDB* enables additional mechanisms like additive Gaussian mechanism to allow existing DP SQL systems to answer more queries with the same level of overall privacy protection under the multi-analyst DP framework. To demonstrate this as a proof-of-concept, we build *DProvDB* over Chorus [28], which is an open-source DP query answering system. That is, *DProvDB* involves Chorus as a sub-module and uses its built-in functions, such as database connection and management, query rewriting (i.e., we use Chorus queries to construct the histogram DP synopses), and privacy accountant (including the basic composition and the Renyi composition). We build our privacy provenance table and implement a query transformation module that can transform the incoming queries into linear queries over the DP views/synopses.

Implementation of the Provenance Table. The privacy provenance table as a matrix can be sparse. Similar to the conventional wisdom applied in access control [15], we can store the privacy provenance table by row or column to reduce the storage overhead.

C.2 Other Results on TPC-H Dataset

We present the missing empirical results on TPC-H dataset, including the end-to-end comparison (Fig. 8), the run time performance

[2] Breeze [25] is a numerical processing and optimization library for Scala.

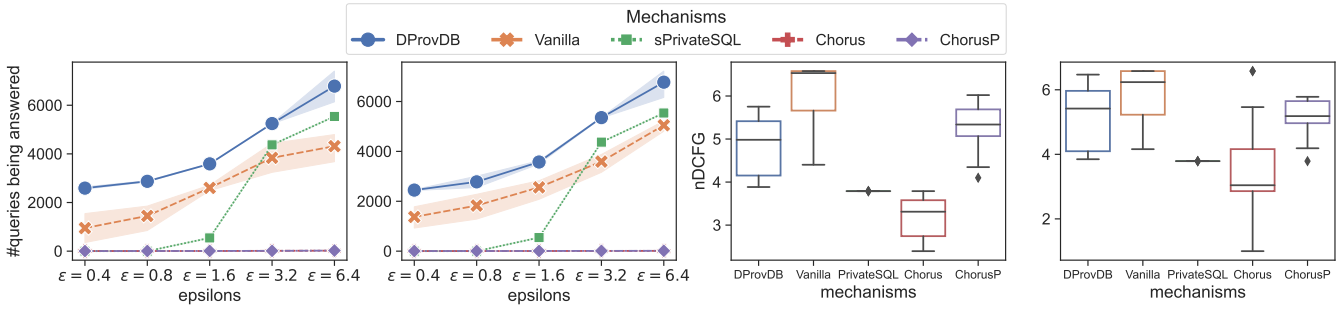

Figure 8: End-to-end Comparison (*RRQ* task, over *Adult* dataset), from left to right: a) utility v.s. overall budget, round-robin; b) utility v.s. overall budget, randomized; c) fairness against baselines, round-robin; d) fairness against baselines, randomized.

Table 3: Runtime Performance Comparison over Different Mechanisms on Adult Dataset (running on a Linux server with 64 GB DDR4 RAM and AMD Ryzen 5 3600 CPU, measured in milliseconds, average over 4 runs)

Systems	Setup Time	Running Time	No. of Queries	Per Query Perf
DProvDB	8051.06 ms	541.77 ms	294.5	1.84 ms
Vanilla	8051.06 ms	393.42 ms	289.5	1.36 ms
sPrivateSQL	8051.06 ms	564.68 ms	295.8	1.91 ms
Chorus	N/A	3583.47 ms	212.0	16.90 ms
ChorusP	N/A	3424.79 ms	212.0	16.15 ms

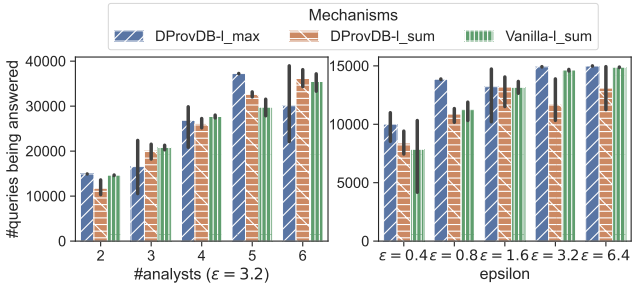

Figure 9: Component Comparison (*RRQ* task, over *Adult* dataset): Additive GM v.s. Vanilla. Left: utility v.s. #analysts, round-robin; Right: utility v.s. overall budgets, round-robin.

comparison (Table 2), and the additive GM component analysis (Fig. 9).

D DESIGN OF VIEWS

As discussed in our problem setup (Section 3), the workload of representative queries are known to the system in advance. Based on the knowledge of this representative query workload, *DProvDB* can properly select a set of materialized views such that each query in the workload can be answered using a single view. Hereby we discuss the design choices for each type of representative SQL queries. In particular, we mainly focus on counting queries and aggregation queries while some of them are not covered by previous work [35].

Handling the GROUP BY Operator. The standard GROUP BY operator in database can leak the information about the activate domain in the database when enforcing differential privacy [40, 42]. Consider the following histogram query that asks the number of employees from each state in a company.

```
SELECT state, COUNT(*)
FROM employee
GROUP BY state
```

By adding noise to the result of this query, the domain knowledge of state is leaked – the adversary can clearly know from which states there are no employees of this company. Existing works [40] modify the GROUP BY operator into GROUP BY* which requires the data analyst to enumerate the domain of the attribute. This GROUP BY* modification does not directly apply to *DProvDB* since *DProvDB* answers queries using DP views. To enable this operator or this type of histogram query, we explicitly bring up the definition of histogram and full domain histogram.

Definition 16. *For a given database instance D and an attribute of the database relation $a_i \in attr(R_i)$, the* histogram *view of this attribute is the 1-way marginal $h(a_i) \in \mathbb{N}^{|Dom(a_i,D)|}$ where each entry of the histogram $h(a_i)[j]$ is the number of elements in the database instance D of value $j \in Dom(a_i, D)$. The* full-domain histogram *view of this attribute a_i is the histogram view built upon the domain of a_i rather than the active domain. That is, $h_f(a_i) \in \mathbb{N}^{|Dom(a_i)|}$, where $h_f(a_i)[j]$ is the number of elements in the database instance D of value $j \in Dom(a_i)$.*

Note that the non-private full-domain histogram view can be sparse, with a number of 0 entries. *DProvDB* generates DP synopses for the full-domain histogram views by injecting independent Gaussian noise with variance σ proportional to $\frac{1}{\epsilon}$ to each entry of the view. Then these DP synopses are able to handle the differentially private GROUP BY operator by either of the following approaches: 1) in compatible with the GROUP BY* operator, *DProvDB* can allow data analyst to enumerate the attribute domain and return the corresponding part (or subset) of the DP synopsis to the data analyst; 2) similar to [54], *DProvDB* can allow data analyst to specify a threshold parameter τ, where the noisy entries of the DP synopsis with counts less than τ will not be returned to the data analyst.

Handling Aggregation Queries. Aggregation queries requires SQL operators including SUM, AVG, MAX, and MIN. This type of queries is not supported by PrivateSQL [35]. A typical running

example of aggregation query is to ask the sum of salaries of all employees in the database, which is shown as follows. The sensitivity of this query equals the maximum value in the domain minus the minimum value in the attribute domain (as we consider the bounded DP).

```
SELECT SUM(salary)
FROM employee
```

There are two main challenges for the DP SQL engine to support this type of queries. First, the DP SQL engine should keep track of the range of the maximum and the minimum value that an attribute can take to correctly calculate the sensitivity of the query. Second, answering the aggregation queries with the simple (full-domain) histogram view incurs large error variances, since the error accumulates when combining the involved bins in the histogram.

To solve the first issue, we incorporate the clipping and rewriting technique [28] into the design of the views. *DProvDB* allows data curator to enforce a clipping lower bound lb and upper bound ub on the domain of the targeting attribute when generating/initializing the views. Then the value of the attribute a_i in the database is regarded as $\max(lb, \min(ub, a_i))$. Therefore, after injecting noise to the histogram (similar to what we described for the GROUP BY operator), the maximum sensitivity of the sum query answered over the histogram is bounded by $(ub - lb)$[3]. We call these generated views as *clipped histogram views*, or clipped views, with bins only in the clipping boundary. Answering the SUM query over the clipped histogram views is just applying the linear combination of the bin value and the bin counts to (a subset of) the histogram bins. Since this clipping operation satisfies the property of constant Lipschitz transformation [6], generating clipped views satisfy (ϵ, δ)-DP, derived by applying for the Lipschitz extension on DP [48].

E FAIRNESS METRICS

We now provide a bit more information on the fairness metrics described in the experimental evaluation part (Section 7). We would like to build algorithms and systems that can achieve fair query answering among data analysts. Inspired by the discounted cumulative gain (DCG) in information retrieval [27, 53], we propose the following discounted cumulative fairness gain (DCFG) metric as the fairness measurement.

Definition 17 (Discounted Cumulative Fairness Gain). *Given n data analysts $A_1, \ldots, A_n$, where the privilege level of the i-th data analyst is denoted by p_i, the fairness scoring of the query-answering of the query engine $\mathcal{M}$ when the overall privacy budget exhausts is calculated as*

$$DCFG_{\mathcal{M}} = \sum_{i=1}^{n} \frac{|Q_{A_i}|}{\log_2(\frac{1}{p_i} + 1)}$$

where $|Q_{A_i}|$ denotes the number of queries answered to the data analyst A_i.

Note that the discounted cumulative fairness gain (DCFG) is proportional to the overall number of queries being answered by the mechanism. If the number of queries being answered is not comparable to each other for different mechanisms, it is not fair to directly compare and justify this DCFG metric. Thus a normalized version of DCFG would be useful in practice.

Definition 18 (Normalized Discounted Cumulative Fairness Gain). *Let $|Q| = \sum_{i=1}^{n} |Q_{A_i}|$ be the total number of queries answered by the system or mechanism to all data analysts from $A_1, \ldots, A_n$: The normalized discounted cumulative fairness gain (nDCFG) is calculated as normalizing DCFG by $|Q|$.*

$$nDCFG_{\mathcal{M}} = \frac{DCFG_{\mathcal{M}}}{|Q|}$$

Example 7. We illustrate the DCFG metric by considering the example where there are three data analysts A_1, A_2, A_3 with privilege level $p_1 = 1, p_2 = 2, p_3 = 4$ (i.e., A_1 has the lowest privilege level while A_3 has the highest privilege level). Supposing there are two mechanisms $\mathcal{M}_1, \mathcal{M}_2$ and the outcomes of them are the following. $\mathcal{M}_1$ answers 10 queries to A_1, 3 queries to A_2 and 0 queries to A_3, whereas $\mathcal{M}_2$ answers 2 queries to A_1, 4 queries to A_2, and 7 queries to A_3. The DCFG score of the first mechanism is calculated as $\frac{10}{1} + \frac{3}{0.585} + \frac{0}{0.3219} = 15.13$ while that of the second mechanism is $\frac{2}{1} + \frac{4}{0.585} + \frac{7}{0.3219} = 30.58$. After normalization, the nDCFG scores of these two mechanisms are 1.16 and 2.35, respectively. Though both mechanisms can answer the same number of queries to the *group of data analysts*, the second one attains higher scores. □

[3]For histogram with domain discretization, the maximum sensitivity is bounded by $\frac{(ub-lb)}{s}$ where s denotes the bin size.