DProvSQL: Accuracy-Aware Privacy Provenance Framework for Differentially Private SQL Engine

by

Shufan Zhang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent years have witnessed the adoption of differential privacy (DP) in practical database query systems. Such systems, like PrivateSQL and FLEX, allow data analysts to query sensitive data while providing a rigorous and provable privacy guarantee. However, existing systems may use more privacy budgets than necessary in certain cases where different data analysts with different privilege levels ask a similar or even the same query. In light of this deficiency, we propose *DProvSQL*, a fine-grained privacy provenance framework that tracks the privacy loss to each single data analyst and we build algorithms that make use of this framework to maximize the number of queries that could be answered. We implement *DProvSQL* as a middleware between the data analysts and the existing differentially private SQL query answering systems. The empirical results on the TPC-H dataset show that our approach can answer around 4x more queries than the baseline approach on average with marginal performance overhead.

Acknowledgements

I would like to thank my supervisor **Prof. Xi He** for all the supportive guidance and professional comments during my entire master's study. I am so fortunate to be advised by her. I appreciate her invaluable feedback all the time, from which I have learnt a lot. This master's thesis is not the end of the journey but just a milestone. I will continue my exploration in the research world, working with her, in the pursuit of the PhD. I also want to thank my thesis committee members, **Prof. Semih Salihoglu** and **Prof. Florian Kerschbaum**, for their time reading my thesis and providing great comments which help improve this work a lot.

I would also like to express my gratitude to my collaborators during the past two years, **Prof. Sharad Mehrotra**, **Prof. David Woodruff**, **Prof. Hongyang Zhang**, **Prof. Haojin Zhu**, **Dr. Primal Pappachan**, and **Dr. Runchao Jiang**. Their expertise, vision, and encouragement motivates my engagement in fun research and inspires me to move forward to become a thinker and researcher, not just a student.

I want to thank my lab mates and DSG members for many inspiring discussions and their support on my research, especially **Christian Covington**, **Karl Knopf**, and **Shubhankar Mohapatra**. They are amazing friends and my senior academic brothers. Lastly, I would like to give special thanks to my personal friends, **Jiayi Chen**, **Xiaohe Duan**, **Zhuangfei Hu**, **Xinda Li**, **Yudong Luo**, **Yanting Miao**, **Peng Shi**, **Liqun Tan**, **Jiaqi Wang**, **Yimu Wang**, **Dake Zhang**, and **Yangyang Zha**, for their help, companion, and support.

People have Dreams. I wish you all the best in pursuing your dreams and finding your very own value. I hope myself would have a wonderful journey in pursuing the PhD at this point of time, too.

Dedication

Dedicated to my parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data collected by companies and organizations can contain sensitive information. With the growing attention on data privacy and the development of privacy protection regulations such as GDPR [49], companies wish to allow external and internal data analysts to make use of the data without compromising the privacy of data contributors. To address the privacy issues, differential privacy (DP) [14] has been considered as a promising and the *de facto* standard for privacy-preserving data analysis these days. In the framework of DP theory, privacy is parameterized by a variable ϵ (or (ϵ, δ) in approximate-DP), called the *privacy budget*, which controls the privacy protection level over the data. A smaller privacy budget indicates larger noise and therefore better privacy protection. Furthermore, the *sensitivity* of a query or a function is used to measure the changing of the results by changing/adding/removing any individual information (i.e., a row) in the database. By carefully injecting controllable noise (e.g., proportional to the privacy budget divided by the sensitivity), researchers or algorithm designers can prove the output of an algorithm (or mechanism, using DP terminology) can only reveal bounded information of any individual in the dataset, and thus this mechanism satisfies the notion of DP.

Recent years have witnessed the adoption of DP from a pure theoretical perspective to practical systems [40] and applications [22, 54]. A plethora of systems are proposed and developed to enforce DP for database management and SQL queries in real-world, including PINQ [36], FLEX [26], PrivateSQL [30], GoogleDP [1], and Chorus [25]. Despite these significant efforts made in existing DP systems, these systems regard the data analysts as a unified entity querying and obtaining the results from the system. Thus the privacy analysis is stark and not personalized as per the data analyst (as shown in Figure 1.1(a)). We argue that data analysts can have different privilege levels in practical scenarios on

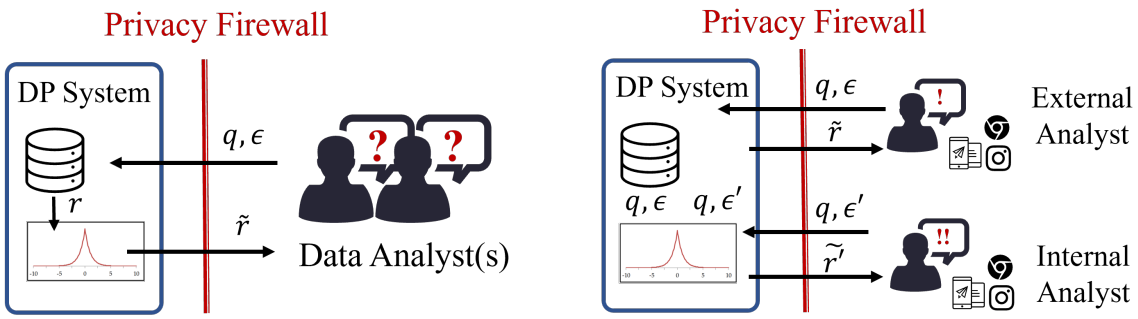

Figure 1.1: (a) Threat Model in Existing DP Systems (e.g., PINQ [36]) (b) Threat Model in the Multi-analysts Motivating Scenarios.

accessing the query results (Figure 1.1(b)), which requires a system to enforce finer-grained privacy tracking. We illustrate this problem using the following motivating example.

Motivating Scenario 1. We consider a protected database (Fig. 1.2) in a corporation that records the data of its employees. This database contains sensitive information about the employees, such as salary and age. Other attributes, like department and state, are less sensitive and considered public information. Two queriers, A, which is an *internal* application in the company, and B, who represents an *external* application outside the company, ask the following aggregation query about the average salary of employees with ages less than 30 for each department:

```
SELECT AVERAGE(salary) FROM employee
WHERE age < 30
GROUP BY department;
```

In this example, the average salary is sensitive information and the query result should be protected with DP. The queriers, application A and application B, cannot access the raw data in the employee table but they are able to learn some noisy answers of the aggregation query. These two applications differ in their privilege of accessing the database and the query results — the internal application (i.e., A) can have a higher privilege level than the external application (i.e., B). In terms of DP, the privacy leakages to A and B through the noisy answers are different, and the internal application A should be able to see more accurate results.

This use case is common in practice for tech companies who need to use sensitive data for internal applications like anomaly detection and also would like to invite external

eid	ename	age	state	salary	department
1	Alice Land	24	AZ	20k	Infrastructure
2	Bobby Hill	28	CA	30k	Infrastructure
3	Carrie Sea	37	WA	40k	Sales
4	Danny Des	26	CA	32k	Cloud

eid	year	role	tax	tax_rate
1	2021	HR	20k	0.15
1	2022	HR	30k	0.25
3	2022	Manager	40k	0.4
4	2020	SDE	32k	0.3

Figure 1.2: The Employee and Tax Table

researchers to analyze their data but with more noise. However, the existing DP systems do not provide tools to distinguish these queriers and track their perspective privacy loss. A naive tracking and answering of each querier's queries independent of the others can waste privacy budgets, i.e., fewer queries can be answered accurately under a given total privacy budget. Additionally, if we assume (all or a subset of) data analysts can communicate with each other and collude, we would like to control the overall privacy loss. This is due to the underlying privacy implication, where the compromised data analysts asking the same query are able to infer a more accurate result of sensitive data, according to the sequential composition theorem of DP [14]. Stated differently, the privacy protection level will degrade when more independent noisy answers to the same query are obtained by the adversary. This challenge to private data management and analytics is mainly on account of the fact that none of the existing DP SQL systems records the individual budget limits 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 the database research [5, 6]. As one can see, without privacy provenance, the query answering process for the multi-analyst use case is not convenient nor secure as those systems are not dedicatedly designed for this real-world application scenario.

Motivating Scenario 2. Existing mainstream DP query answering systems, such as PINQ [36], Chorus [25], and PrivateSQL [30], allow the user or querier to write differentially private programs or specify the privacy budget with SQL-like queries. These systems do not provide any (optimality) guarantees on the utility/accuracy of the queries. Thus, to obtain the desired utility, the queriers are required to know well about differentially private mechanisms and how to appropriately set the privacy budget for their queries. This deficiency motivates researchers to investigate and design *accuracy-aware* systems [33, 38, 34], which allow users to specify accuracy requirements instead of privacy budget and (the system) can automatically translate the accuracy requirements to the minimal privacy budget. However, it is not clear and trivial to see how the accuracy-aware module can be adapted to the system supporting multiple data analysts (with different privilege levels) while answering

as many queries as possible when data analysts collude. Furthermore, while there exist accuracy-aware DP systems proposed in very recent years (like APEx [18] and MIDE [19]) for data exploration and analytics purposes, these systems are limited to only answering linear (counting) queries, which is insufficient for a full-fledged query answering system. To better illustrate this problem, we consider the following join query as an example:

```
SELECT department, AVERAGE(tax_rate)
FROM employee JOIN tax
ON employee.eid = tax.eid
WHERE age > 28
GROUP BY department;
```

This query expresses "what is the average tax rate for employees older than 28 in each department" and is a join query between the employee table and the tax table. The global sensitivity of a join query is unbounded, because a join query can multiply input records from different tables and therefore adding or removing one row in a table can affect (possibly) unbounded numbers of rows in the output of the query. In the example above, an employee can have multiple tax records recorded on the tax slip (or the tax table). The state-of-the-art approaches [41, 26, 11] to answer join queries with differential privacy guarantee is to consider the instance-specific sensitivity and inject noise proportional to it, where in addition, [10, 11] show their approaches can achieve instance-optimal noise injection. However, since all these approaches are dependent on the specific database instance, there are no general accuracy/utility guarantees on answering a join query, which is challenging to appropriately translate a given accuracy requirement into the minimal privacy budget.

1.1 Contributions

To tackle with these challenges, we propose *DProvSQL*, a new privacy provenance framework for differentially private SQL engine that fits in the multi-analysts scenario. Instead of answering queries from each data analyst independently, *DProvSQL* generates DP synopses for a set of views, so that the queries can be answered based on these synopses and these synopses can be dynamically updated according to data analysts' requests. Furthermore, *DProvSQL* enables a privacy provenance table that enforces a fine-grained privacy provenance 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, we build an additive Gaussian mechanism, maintaining global (viz., as per view) and local (viz., as per analyst) DP synopses (i.e., materialized results for views) to answer as many queries presented to the system as possible. The ongoing work of this project aims at extending our system to not only answer queries where the global sensitivity is bounded, but also the join queries over the multi-relational database. We implement *DProvSQL* as a middleware between the data analysts and the existing DP SQL query system, providing the aforementioned nice functionalities to such a system. Our empirical results show a significant improvement over the baseline method where queries are answered independently, in terms of the number of queries being answered and the minimum expected error among the answers. The technical contributions of this work are highlighted as follows.

- **New DP framework** with intriguing research questions. We propose the multi-analyst DP framework where mechanisms satisfying multi-analyst DP provide discrepant answers to analysts with different privilege levels. We study the privacy analysis, budget allocation, and fair query answering problems under this setting.
- **New end-to-end architecture** providing fine-grained privacy tracing for DP. We propose the privacy provenance table that can track the consumed privacy budget as per analyst and as per view. The intrinsic privacy constraints enforced on this table enable dynamic budget allocation and fair query answering.
- **New DP mechanism** that provides tight composition bound when analysts collude. We propose global/local synopses and design the additive Gaussian mechanism that injects correlated noise to generate local synopses from global synopses to achieve the tight collusion bound among analysts.
- **New query answering system** implemented as a middleware between the multiple data analysts and existing DP query systems. Empirical evaluation shows the efficacy and efficiency of our system.

1.2 Paper Roadmap

The remainder of this thesis is outlined as follows. Chapter 2 introduces the necessary notations and background knowledge on database and differential privacy. Our multi-analysts DP query answering research problems are formulated in chapter 3 and a high-level overview of our proposed system is briefed in chapter 4. Chapter 5 describes the

details of our design of the DP mechanisms and system modules. In chapter 6, we present the system implementation specifics and an empirical evaluation of our system with the baseline approaches. In chapter 7 we go through the related works and we discuss the potential extension and directions for ongoing and future work in chapter 8. We conclude this work in chapter 9.

Chapter 2

Preliminaries

We consider the database instance D that stores sensitive data with a set of schema/relations $\mathcal{R} = \{R_1, \ldots, R_l\}$. The domain of all database instances is denoted by $\mathcal{D}$. Each relation $R_i \in \mathcal{R}$ consists of a set of attributes, $attr(R_i) = \{T_1, \ldots, T_j\}$. We denote the domain of an attribute T_j by $Dom(T_j)$ while $|Dom(T_j)|$ denotes the domain size of that attribute. Additionally, we use $Dom(T_j, D)$ and $|Dom(T_j, D)|$ to denote the active domain, which is the actual domain of this attribute in a specific database instance D, and the size of the active domain of the attribute T_j. We introduce and summarize the related definitions of differential privacy as follows.

Definition 1 (Neighbouring Databases)**.** *We say two databases D and D' are neighbouring databases if D and D' differ in at most one tuple, i.e., $D = D' \backslash \{r\}$ or $D = D' \cup \{r'\}$.*

Definition 2 (Differential Privacy (DP))**.** *We say that a randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{O}$ satisfies (ϵ, δ)-differential privacy, if for any two neighbouring databases D and D' that differ in only 1 tuple, and all $O \subseteq \mathcal{O}$, we have*

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

where the probability is taken over the randomness used by the mechanism $\mathcal{M}$.

DP enjoys many useful and nice properties, for example, post-processing and sequential composition [14].

Theorem 1 (Post Processing [14])**.** *For any mechanism $\mathcal{M}$ that achieves (ϵ, δ)-DP, applying any arbitrary randomized function f over the output of $\mathcal{M}$, that is, the composed mechanism $f \circ \mathcal{M}$, satisfies (ϵ, δ)-DP.*

Table 2.1: Notation Cheatsheet

Notation	*Definition*
$\mathcal{D}, D$	Database domain, a specific database instance
Ψ, ψ	(A set of) Privacy constraint(s)
$\epsilon(, \delta)$	The privacy budget
$\mathcal{A}, A_i$	(A set of) Data analyst(s)
$\mathcal{R}, R_i$	(A set of) Schema(s)/Relation(s)
$\mathcal{R}_{Priv}, \mathcal{R}_{Pub}$	Set of private/public relations
$Dom(T_i)$	Domain of the attribute T_j
$\|Dom(T_i)\|$	Size of the domain of the attribute T_j
$\mathcal{M}$	DP mechanism
P	The privacy provenance table
$\mathcal{V}, V$	(A set of) View(s)
$V^{\epsilon}, V^{\epsilon}_{A_i}$	Global/Local synopsis
$GS(\Delta q), LS$	Global/Local sensitivity
$LS^{(k)}_q$	Local sensitivity of query q at distance k
$\hat{SS}, SS$	(Upper bound on) The smooth sensitivity

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.

Theorem 2 (Sequential Composition [14])**.** *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.*

The sequential composition is trying to bound 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 proofs of theorem 1 and theorem 2 can be seen in [14].

Definition 3 (l_2-Global Sensitivity)**.** *For a function $q : \mathcal{D} \to \mathbb{R}^d$ and all $D, D' \in \mathcal{D}$, the l_2 global sensitivity of this function is defined as*

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

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

Definition 4 (Gaussian Mechanism [14]). *Let $\epsilon \in (0,1)$. Given a numerical query $f : \mathcal{D} \to \mathbb{R}^d$, for constant $c > \sqrt{2\ln(1.25/\delta)}$, the Gaussian mechanism adds the noise vector $(\eta_1, \eta_2, \ldots, \eta^d)$ to the query answer $f(D)$, where η_i are i.i.d. random variables drawn from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma > c\Delta f/\epsilon$. The Gaussian mechanism is (ϵ, δ)-differentially private.*

The standard Gaussian mechanism [14] has the limitation that it can only be used in high privacy regime, where the privacy parameter ϵ should be within the range of $(0, 1)$. Balle and Wang [2] propose an improved mechanism, namely the analytic Gaussian mechanism, overcoming this limitation in standard Gaussian mechanism.

Definition 5 (Analytic Gaussian Mechanism [2]). *Let $q : \mathcal{D} \to \mathbb{R}^d$ be an arbitrary d-dimensional function. The analytic Gaussian mechanism $\mathcal{M}(D) = q(D) + \eta$ where $\eta \sim \mathcal{N}(0, \sigma^2)$ 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 the DP framework, a mechanism that achieves DP adds noise to the query result, which involves errors in the answer. We measure the *data utility* of the query answer using the expected squared error, defined as follows.

Definition 6 (Data Utility). *For a query $q : \mathcal{D} \to \mathbb{R}^d$ and a mechanism $\mathcal{M} : \mathcal{D} \to \mathbb{R}^d$, the data utility of mechanism $\mathcal{M}$ is measured as the expected squared error, $v = \mathbb{E}[q(D) - \mathcal{M}(D)]^2$. For the (analytic) Gaussian mechanism, the expected squared error equals to its variance, that is, $v = \sigma^2$.*

Enforcing DP on a complex database SQL query may require the analysis of a set of aggregations. PINQ [36] proposes the concept of stable transformations, which is a useful notion to bound the DP privacy implication through a set of transformations/operations on the database.

Definition 7 (c-Global Stability [36]). *We say a transformation $T : \mathcal{D} \to \mathcal{D}$ satisfies c-global stability if, $\forall$ two input databases $D, D' \in \mathcal{D}$,*

$$|T(D) \triangle T(D')| \leq c \times |D \triangle D'|,$$

where $\triangle$ denotes the symmetric difference between two databases, i.e., $D \triangle D' = (D \backslash D') \cup (D' \backslash D)$.

For example, common SQL transformations SELECT, PROJECT, and COUNT have a stability of 1 and GROUP BY has a stability of 2. Stability is a useful definition to bound the differential privacy guarantee for a sequence of transformations. [36] proves that for a ϵ-differentially private mechanism $\mathcal{M}$ and a c-stable transformation T, the composition $\mathcal{M} \circ T$ is $(c \cdot \epsilon)$-differentially private.

Chapter 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 data curator 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. In addition, in our multi-analysts model, data analysts may submit the same query and collude to leak more information about the sensitive data.

Differing from prior work [25, 30], these analysts have different privilege levels. We would like to define the privacy per analyst provenance framework as a DP variant that guarantees different levels of privacy loss to the analysts.

Definition 8 (Multi-analyst DP)**.** *We say a randomized mechanism $\mathcal{M} : \mathcal{D} \to (\mathcal{O}_1, \ldots, \mathcal{O}_m)$ satisfies $[(A_1, \epsilon_1, \delta_1), ..., (A_m, \epsilon_m, \delta_m)]$-multi-analyst-DP if for any two databases D and D' that differ in only 1 tuple, any $i \in [m]$, and all $O_i \subseteq \mathcal{O}_i$, 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 output released to the ith analyst.

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

Theorem 3 (Sequential Composition)**.** *Given two randomized mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1 : \mathcal{D} \to (\mathcal{O}_1, \ldots, \mathcal{O}_m)$ satisfies $[(A_1, \epsilon_1, \delta_1), ..., (A_m, \epsilon_m, \delta_m)]$-multi-analyst-DP, and $\mathcal{M}_2 : \mathcal{D} \to (\mathcal{O}'_1, \ldots, \mathcal{O}'_m)$ 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) to each coordinate, we can get this composition upper bound for multi-analyst DP. □

Comparison to Personalized DP [27, 16]. Some exiting works consider the personalized DP (PDP) framework which provides discrepant privacy protection guarantees to different *data contributors in the protected database*. Our multi-analyst DP framework is dual to theirs, where each data analyst has discrepant privacy privileges, referring to their allowed maximum privacy budgets. Under this new multi-analyst DP framework, several research questions are raised and thereby motivate our work. We justify them as follows.

RQ 1: worst-case privacy analysis across analysts. If the data analysts do not collude, we can use sequential composition (Theorem 3) to track the privacy loss to each data analyst for all queries this analyst asks. However, if all or a subset of data analysts collude or are compromised by an adversary, how to design algorithms to account the privacy loss to the colluded analysts?

Assuming a query is asked once by all data analysts, when all the analysts are compromised by an adversary, the privacy loss to this adversary is upper bounded by $(\sum \epsilon_i, \sum \delta_i)$, suggested by the sequential composition theorem, and it is lower bounded by $(\max \epsilon_i, \max \delta_i)$, where (ϵ_i, δ_i) is the privacy loss to the ith analyst.

RQ 2: dynamic budget allocation across views. Prior works for DP query answering [30, 20] often assume the availability of a representative workload that can capture the queries which would be popular among the data analysts in the future. Given such a representative workload, a system [30] can select a set of views $V_1, V_2, \ldots, V_l$ such that each query in this workload can be answered with a linear query on a single view, and then generate a DP synopsis for each of the selected views. Given a total privacy budget, prior work [30, 35] splits the privacy budget equally or proportional to the sensitivity of the view to achieve an equal accuracy rate. However, some views may require higher accuracy than others, depending on the requests of the data analysts. Therefore, it is important to design an algorithm that can dynamically allocate privacy budgets to the given views and update their corresponding DP synopses over time. In this work, we consider the histogram view, which is defined as follows.

Definition 9 (Histogram View)**.** *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 $j \in Dom(a_i)$.*

This histogram view can be naturally extended to the view generated over multiple attributes. Such a view is called a contingency table. Given a view, we consider the query answerability and transformation as in related work [30].

Definition 10 (View and Query Answerability [30])**.** *Given a database instance D, a materialized view $\mathbf{V}(D)$ (or V) is a set of results of counting query about some specific domain values over some attributes in the database instance. For a query q over the database instance D, if there exists a query q' over the histogram view V such that $q(D) = q(\mathbf{V}(D))$, we say the query q is answerable over the view V.*

RQ 3: fair query answering among data analysts. Under standard DP with no distinction among analysts, a data analyst can ask any number of queries as long as the composition of these queries does not exceed the overall system privacy budget. Letting a data analyst with a low privilege level consume too much privacy budget is unfair to those with higher privacy privileges. The privilege level in this context refers to the maximum allowed privacy budget of the data analyst. 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 [24, 50], we propose the following discounted cumulative fairness gain (DCFG) metric as the fairness measurement.

Definition 11 (Discounted Cumulative Fairness Gain (DCFG))**.** *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 when the overall privacy budget exhausts is calculated as*

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

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

Example 1. *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 $\frac{2}{1} + \frac{4}{0.585} + \frac{7}{0.3219} = 30.58$. Though both mechanisms can answer the same number of queries to the group of data analysts, the second mechanism can achieve higher fairness scores.

The main focus of this thesis is on research questions 1 and 2. While our solution can avoid spending more privacy budgets on data analysts with low privilege levels, we do not incorporate fairness considerations into the algorithm design in this work. In ongoing and future work, we will look into building a fair mechanism that provides clear fairness guarantees.

Chapter 4

System overview

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

4.1 Key Design Principles

To support the multi-analysts use case and to answer the aforementioned research questions, we identify the following four principles for a differentially private SQL query system and propose a system *DProvSQL* that follows these principles.

Principle 1: fine-grained privacy provenance. The 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 differentially private views/synopses in the system. Compared to directly answering a query from the database D, view-based query answering can answer more differentially private queries [30], but it assumes the accessibility of a pre-known query workload. In our system, 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 privacy budget associated with their query, the system enables a second accuracy-aware

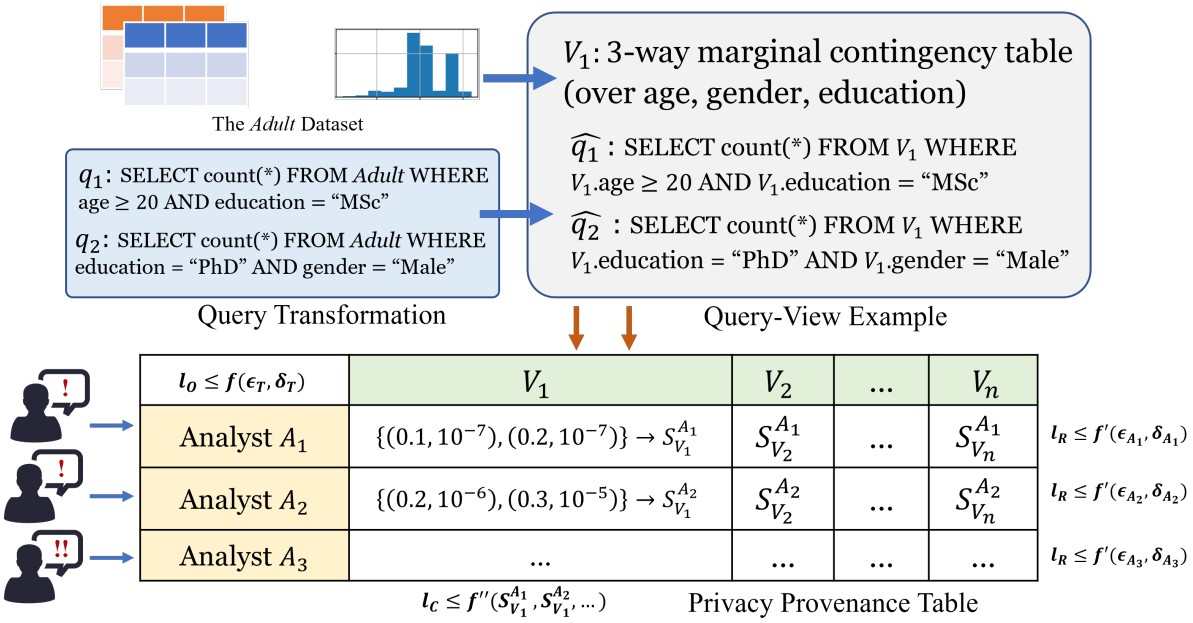

Figure 4.1: The Privacy Provenance Table: Data Structure

mode. That is, with this mode, data analysts can submit the query with their desired accuracy level in terms of the expected squared error. The dual mode system tolerates data analysts from domain experts, who can take full advantage of their privacy 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 as possible, without violating the privacy constraint specified by the data curators as per data analyst and per view based on their privilege levels.

Supported Queries. *DProvSQL* supports linear counting query, histogram query (GROUP BY operator), aggregation queries (i.e, SUM, AVG, MAX, and MIN), and we can extend *DProvSQL* to answer join queries over multi-relational databases.

4.2 Privacy Provenance Table

To meet the first two principles, we propose a privacy provenance table for *DProvSQL*, which is inspired by the access matrix model in access control literature [48], 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 $(\mathcal{A}, \mathcal{V}, P)$, 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 P the privacy provenance table, defined as follows.

Definition 12 (Privacy Provenance Table)**.** *The privacy provenance table P is a matrix that tracks the privacy loss of the database as per each data analyst in $\mathcal{A}$ and each query-view in $\mathcal{V}$. Each row of P corresponds to a data analyst and each column of P corresponds to a query-view. Each entry of the matrix $P[A_i, V_j]$. records the current cumulative privacy loss, on view V_j to analyst A_i.*

The system administrator can specify different levels of privacy constraints over the privacy provenance table, defined as follows. Different data analysts may have different privilege levels, where this difference reflects in the allowed maximum privacy budget for the analysts. This is expressed as the row-level constraint enforced on this privacy provenance table.

Definition 13 (Privacy Constraints)**.** *The table also includes a set of row/column/table privacy constraints, Ψ. A* row constraint *for ith row, denoted by ψ_{A_i}, refers to the total privacy loss to a particular data analyst A_i (according to his/her privilege level) while a* column constraint *for the jth column, denoted by ψ_{V_j} refers to as the allowed maximum privacy loss to a specific view V_j. We use the* table constraint *over P, denoted by ψ_P, to specify the overall privacy loss that is allowed for the protected database.*

The privacy constraints can be correlated. The internal restriction over these constraints indicates that the privacy loss in each entry cannot exceed row and column constraints while the row/column constraints cannot exceed the overall table constraint. We can therefore use these constraints to enforce sanity checking to decide whether issue or reject a query from a data analyst.

Example 2. *Figure 4.1 shows an example of the privacy provenance table. We consider a histogram view V_1 is a 3-way contingency table over attributes (age, gender, and education). The queries q_1 and q_2 can be transformed into $\hat{q}_1$ and $\hat{q}_2$ that are answerable using this histogram view V_1. Three data analysts A_1, A_2 and A_3 with different privilege levels are recorded in privacy provenance table and we track every privacy budget spent over time on the views.*

To meet the fourth principle, we formulate the *maximum query answering problem* based on the privacy provenance table.

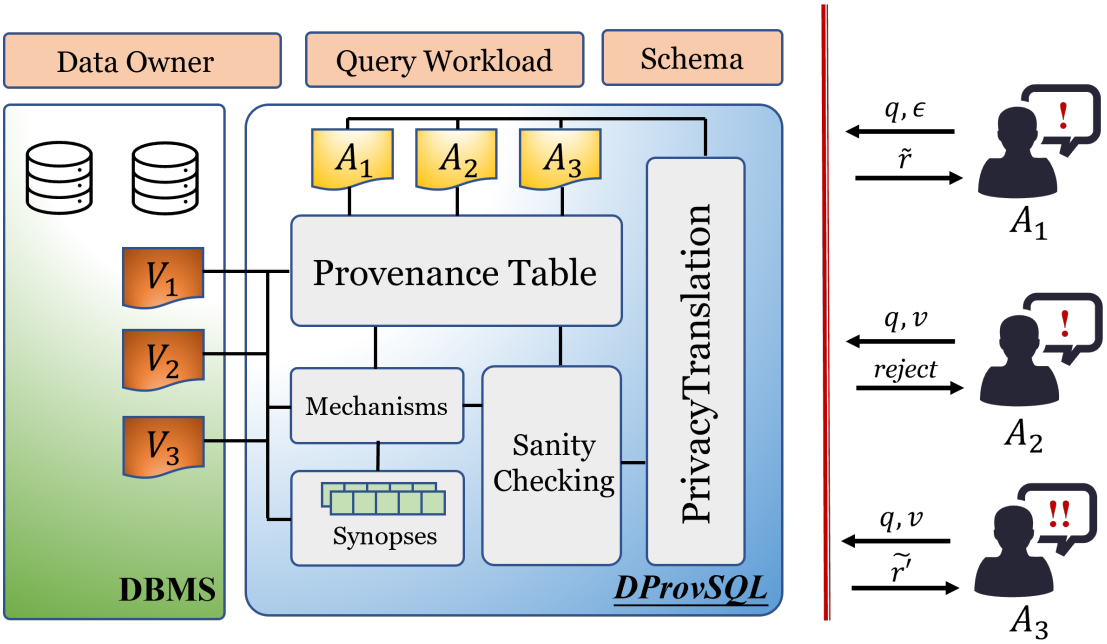

Figure 4.2: The System Architecture of *DProvSQL*

Problem 1. *Given a privacy provenance table $(\mathcal{A}, \mathcal{V}, P)$, 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 *DProvSQL* Architecture

Figure 4.2 demonstrates the system architecture of *DProvSQL*. The *DProvSQL* system works as a middle-ware between data analysts and existing DP DBMS systems (such as PINQ, Chorus, and 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. *DProvSQL* maintains the privacy provenance table as introduced in Section 4.2. The privacy provenance table maintains an entry for each registered data analyst and each generated view in the DBMS. With the aid of the privacy provenance table, *DProvSQL* performs sanity checking to detect if any privacy constraint is

violated and thereby decide if the incoming query from a data analyst should be answered or rejected. We further build DP mechanisms to maintain and update the DP synopses and the privacy provenance table.

Dual Query Submission Mode. *DProvSQL* provides two types of query submission modes or interfaces to data analysts. For data analysts who are DP experts with the knowledge of optimally apportioning privacy budgets across different queries [26], the *DProvSQL* enables the **privacy-oriented mode**. Similar to prior work in DP query answering [36, 26, 25, 55], this mode allows the data analysts to specify a privacy budget associated with their query for answering that query. In this mode, the goal of *DProvSQL* is to find a mechanism that can optimize the data utility of the query result (i.e., minimize the expected squared error) for the given privacy budget ϵ, δ. On the other hand, *DProvSQL* enables the **accuracy-oriented mode**, where the data analysts can attach a utility measure (i.e., maximum expected squared error) with the submitted query. *DProvSQL* is tuned to minimize the usage of the privacy budget while answering the query with the required utility. If the remaining privacy budget for this data analyst, as recorded in privacy provenance table, is not sufficient for satisfying the utility requirement, the query will be rejected. This type of mode is more user-friendly to DP novice data analysts, and hence, has been gradually adopted by more recent DP query batch processing systems such as APEx [18] and DPella [34].

Algorithm Overview. Algorithm 1 summarizes how *DProvSQL* uses the DP synopses to answer incoming queries. At the system setup phase (line 1-2), the system (or data curator) initializes the privacy provenance table by setting the privacy budget as per entry as 0 and the row/column/table constraints Ψ. The system initializes empty global/local 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, *DProvSQL* checks if answering q_i with budget ϵ_i will violate the privacy constraints Ψ (Line 8). If this sanity check passes, we run the mechanism to obtain a noisy synopsis (line 9). *DProvSQL* uses this synopsis to answer query q_i and returns the answer to the data analyst (line 10). If the sanity check fails, *DProvSQL* rejects the query (line 12).

Algorithm 1: System Overview

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

Data curator sets up the privacy provenance table P with row/column/table constraints Ψ.
Initialize all the synopses for $V \in \mathcal{V}$
repeat
    Receive (q_i, v_i) from data analyst A_i
    Select view $V \in \mathcal{V}$ to answer query q_i
    Select mechanism $M \in \mathcal{M}$ applicable to q_i
    $\epsilon_i \leftarrow$ M.privacyTranslate(q_i, v_i, V)
    if *M.constraintCheck(* $P, A_i, V, \epsilon_i, \Psi$ *)* **then**
        $V_{A_i}^{\epsilon'} \leftarrow$M.run$(P, A_i, V, \epsilon_i)$
        Answer q_i with $V_{A_i}^{\epsilon'}$ and return answer r_i to A_i
    else
        reject the query q_i
    end
until *No more queries sent by analysts*

Chapter 5

DP Algorithm Design

Algorithm 1 outlines the abstract interface (with key components) of our system. In this chapter, we design concrete differentially private algorithms that can fit into this framework, assuming that the system is running over a single-relation database. In particular, we first describe a baseline DP mechanism that can instantiate the system interface but cannot maximize the number of queries being answered. Then we propose additive Gaussian mechanism that leverages the correlated noise in query answering to improve the poor utility of the baseline mechanism. We only consider queries with global sensitivity that is bounded in this section.

5.1 Baseline Approach

The baseline approach is based on the standard usage of the Gaussian mechanism (applied to both the basic Gaussian mechanism [14] and the analytic Gaussian mechanism [2]). We describe how the system modules, i.e., the privacy translation, the DP constraint enforcement, and the privacy provenance table maintenance, are instantiated with the baseline approach.

5.1.1 Accuracy-Privacy Translation

A key module for the *accuracy-oriented mode* in *DProvSQL* is a *accuracy-privacy translator* that interprets the user-specified utility requirement into the minimum privacy budget. The consumed privacy budget on this query will be recorded in the privacy provenance

Algorithm 2: Baseline Approach

Set δ in the system
Function *run*(P, A_i, V, ϵ_i) **:**
    Generate a synopsis $V_{A_i}^{\epsilon_i}$ from view V
    Update privacy provenance table $P[A_i, V]$ by $P[A_i, V] \leftarrow P[A_i, V] + \epsilon_i$
    return $r_i \leftarrow V_{A_i}^{\epsilon_i}$
end
Function *privacyTranslate*(q_i, v_i, V) **:**
    candidateEps = [ ]
    $v \leftarrow$ CALCULATEVARIANCE(q_i, v_i, V)
    for $\epsilon \leftarrow \epsilon_{max}, \epsilon_{max}/2, \ldots, \epsilon_{max}/2^{10}, 0$ **do**
        if $\Phi_{\mathcal{N}}\left(\frac{\Delta q_i}{2v} - \frac{\epsilon v}{\Delta q_i}\right) - e^{\epsilon}\Phi_{\mathcal{N}}\left(-\frac{\Delta q_i}{2v} - \frac{\epsilon v}{\Delta q_i}\right) \leq \delta$ **then**
            candidateEps.ADD(ϵ)
    end
    return candidateEps.MIN()
end
Function *constraintCheck*$(P, A_i, V_j, \epsilon_i, \Psi)$ **:**
    if $P.composite() + \epsilon_i > \Psi.\psi_P$ **then**
        return False
    if $P.composite(axis{=}Row) + \epsilon_i > \Psi.\psi_{A_i}$ **then**
        return False
    if $P.composite(axis{=}Column) + \epsilon_i > \Psi.\psi_{V_j}$ **then**
        return False
    return True
end

table and returned with the query result to the data analysts, if answering this query does not violate the constraint specifications in the privacy provenance table. In the baseline mechanism, we use the following analytic Gaussian translation algorithm, defined as an optimization problem in Definition 14, which 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 14 (Analytic Gaussian Translation)**.** *Given a numerical query $q : \mathcal{D} \to \mathbb{R}^d$ with sensitivity Δq, in order to achieve (ϵ, δ)-DP and a total expected squared error bound v for this query, the outputting privacy budget should satisfy*

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

That is, given $\Delta q, \delta, v$, to solve the following optimization problem to find the minimal ϵ.

$$\underset{\epsilon}{\text{minimize}} \quad \epsilon \tag{5.1}$$

$$\textit{subject to} \quad \Phi_{\mathcal{N}}\left(\tfrac{\Delta q}{2v} - \tfrac{\epsilon v}{\Delta q}\right) - e^{\epsilon}\Phi_{\mathcal{N}}\left(-\tfrac{\Delta q}{2v} - \tfrac{\epsilon v}{\Delta q}\right) \leq \delta \tag{5.2}$$

$$\epsilon \in (0, \epsilon_{max}]. \tag{5.3}$$

We note that, given the error bound v (and $\Delta q, \delta$), it is not easy to find a closed-form solution for solving and finding the minimum ϵ to solve the optimization problem in Definition 14. If we regard the LHS of equation 5.2 as a function of ϵ, this function is known as *monotonically decreasing* with respect to ϵ [2]. Therefore, we can use a binary line search algorithm (Algorithm 2: 7-14) to find the minimal ϵ. Specifically, we start from the maximal ϵ and in each iteration we halve the privacy budget and test if equation 5.2 satisfies. We thus can find an approximately minimal ϵ with the constraints. This translation can be done offline[1] and we can store the (ϵ, v) mapping into a translation look-up table in the system for later run-time queries. Additionally, we should note that we cannot directly use the accuracy bound the data analyst specified over the query as the error bound in equation 5.2. This is because, instead of adding noise to the query result, we add noise to the view and then answer the query based on the noisy synopsis. Answering the query over the histogram synopsis may require adding up bins which will scale up the noise variance. Therefore, we need to look into the query and the view structure to calculate the error bound (line 9 in Algorithm 2) and then run the search algorithm.

[1]The offline pre-computing of a accuracy-privacy translation table would lose some precision in translating the accuracy requirement, because we need to fix some accuracy step size during the pre-computing.

5.1.2 Provenance Sanity Checking

As mentioned, the data curator can specify privacy constraints over the privacy provenance table $\Psi = \{\psi_{A_i} | A_i \in \mathcal{A}\} \cup \{\psi_{V_j} | V_j \in \mathcal{V}\} \cup \{\psi_P\}$, which consist of the per analyst constraints, per view constraints, and the table constraint. *DProvSQL* uses the provenance sanity checking method to decide to whether *reject or issue a query* to an analyst based on the privacy provenance table and these constraints. The constraint checking algorithm in the baseline mechanism is simple and intuitive. As shown in Algorithm 2: 16-23 (the function CONSTRAINTCHECK), it checks that, if the current query was to issue, whether the table constraint (line 20-21), the row constraint (line 22-23), and the column constraint (line 24-25) would be violated. If any one of them is violated, *DProvSQL* rejects the query; otherwise it runs the query answering process. The COMPOSITE function in this constraint checking algorithm can refer to the basic sequential composition, or tighter privacy composition given by Renyi-DP [37] or zCDP [15, 4], depending on the underlying system [25].

The constraints specified on the privacy provenance table are internally correlated: 1) the privacy budget recorded in each entry cannot exceed the ones specified in the corresponding row constraint (i.e., analyst constraint) and the column constraint (i.e., view constraint); 2) either the row constraint or the column constraint cannot exceed the table constraint. These intrinsic correlations among the constraints provide nice properties in subsuming existing privacy budget allocation mechanisms (e.g. as in PrivateSQL [30]) and solving our dynamic view budget allocation problem (RQ2) in Section 3.

First, according to the correlations, the table constraint can be set to a constant or, automatically, a function of row constraints and column constraints $\psi_p = \min(\sum_i \psi_{A_i}, \sum_j \psi_{V_j})$. To achieve the existing privacy budget allocation scheme in [30], the data curator can follow the *query fair allocation strategy* (or other advanced static strategies) and specify the view constraints as $\{\psi_{V_j} | V_j \in \mathcal{V}\} = \{\lambda_{V_j} \cdot \epsilon / \hat{\Delta}_{V_j}\}_{\forall V_j \in \mathcal{V}}$, where $\hat{\Delta}_{V_j}$ is the upper bound of the sensitivity of the view V_j [30]. In correspondence to the dynamic view budget allocation problem, the data curator can use *DProvSQL* to set the table constraint to be a constant and then the view constraints as ∞ (this basically means the view constraint checking will always be bypassing). Then *DProvSQL* can issue or reject the query based only on the analyst constraints and the table constraint. More privacy budget can be assigned on a view that the corresponding queries are more frequently being asked.

We note that the privacy provenance table as a matrix can be sparse. Similar to the conventional wisdom applied in access control, we can store the privacy provenance table by row or column to reduce the storage consumption.

5.1.3 Putting Components All Together

The baseline approach is aligned with existing DP SQL query systems in the sense that we add independent noise to the result of each query. That being said, we can easily adopt this baseline approach as a middle-ware to existing systems to provide the privacy provenance and accuracy-aware features, with the least amount of effort of changing existing systems. The way this baseline method works is outlined in Algorithm 2: 2-5 (the function RUN). We simply generate the differentially private synopsis $V_{A_i}^{\epsilon_i}$ using analytic Gaussian mechanism from the view V and update the corresponding entry $P[A_i, V]$ in the privacy provenance table by adding up the consumed privacy loss ϵ_i on the query. This baseline mechanism satisfies the following accuracy and privacy properties.

Theorem 4. *Given a query q where its global sensitivity is bounded, the baseline mechanism (Algorithm 2) returns the query result with the expected squared error at most v and satisfies differential privacy with a minimal cost of privacyTranslate(q, v).ϵ. The privacy guarantee of Algorithm 2 follows that of Theorem 7.*

Remark. Without loss of generality, we can assume that 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). If we were to relax this assumption in the baseline approach, we need to store all historical synopses to avoid spending additional budgets on answering the decreasing accuracy queries. We will described how this issue could be better solved with additive Gaussian mechanism.

5.2 Additive Gaussian Approach

DP synopses are differentially private query results of views executed on a database instance. We introduce a new additive Gaussian mechanism which is used many time in our synopses maintenance, and then describe how *DProvSQL* generates and updates the (local and global) DP synopses with this algorithm design.

5.2.1 Additive Gaussian Mechanism

We first introduce a simple modification to the standard Gaussian mechanism, named additive Gaussian mechanism (additive GM), that makes use of the nice statistical property

Algorithm 3: Additive Gaussian Noise Calibration

Input: Analysts $\mathcal{A} \coloneqq A_1, \ldots, A_n$; A query q; Database instance D; A set of privacy budgets $\mathcal{B} \coloneqq (\epsilon_1, \delta), (\epsilon_2, \delta), \ldots, (\epsilon_n, \delta)$.

Output: A set of noisy answers $r_1, r_2, \ldots, r_n$.

Function *additiveGM*$(\mathcal{A}, \mathcal{B}, q, D)$ **:**

- $r \leftarrow \text{QUERYEXEC}(q, D)$ ▷ Obtain true query answer.
- $\Delta q \leftarrow \text{SENSCALC}(q)$ ▷ Sensitivity calculation.
- $\mathcal{B}' \leftarrow \text{SORT}(\mathcal{B}, \epsilon_i)$ ▷ Sort $\mathcal{B}$ on the desc order of ϵ's.
- $(\epsilon_i, \delta) \leftarrow \text{POP}(\mathcal{B}')$ ▷ Pop the 1st element.
- $\sigma_i \leftarrow \text{ANALYTICGM}(\epsilon_i, \delta, \Delta q)$ ▷ Refer to [2]
- $r_i \leftarrow r + \eta_i \sim \mathcal{N}(0, \sigma_i^2)$ ▷ Add Gaussian noise.
- **while** $\mathcal{B}' \neq \varnothing$ **do**
  - $(\epsilon_j, \delta) \leftarrow \text{POP}(\mathcal{B}')$
  - $\sigma_j \leftarrow \text{ANALYTICGM}(\epsilon_j, \delta, \Delta q)$ ▷ Refer to [2]
  - $r_j \leftarrow r_i + \eta_j \sim \mathcal{N}(0, \sigma_j^2 - \sigma_i^2)$;
- **end**
- **return** $\mathcal{R} \coloneqq \{r_i | i \in [n]\}$;

end

of the Gaussian distribution. We will later show how this mechanism is used in maintaining the synopses.

It is well known in probability theory that the sum of i.i.d. normal random variables is still normal distributed. We build the *additive Gaussian mechanism* (additive GM) primitive by making use of this fact. The details of additive GM are outlined 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 that satisfy their corresponding privacy budget. The key idea of this primitive is to only execute the query (to get the true answer on the database) only 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.

Theorem 5. *Given a database D, a set of privacy budgets $\mathcal{B} \coloneqq (\epsilon_1, \delta), (\epsilon_2, \delta), \ldots, (\epsilon_n, \delta)$*

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 w.r.t their privacy budget satisfies $[(A_1, \epsilon_1, \delta), ..., (A_n, \epsilon_n, \delta)]$-multi-analyst-DP and $(\max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \delta)$-differential privacy.

Proof. 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,

$$\Pr[\mathcal{M}_{aGM}(D) = r_j] \leq e^{\epsilon_j} \Pr[\mathcal{M}_{aGM}(D') = r_j] + \delta_j. \tag{5.4}$$

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 5.4 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 $r_j \coloneqq r_i + \eta_j = r + \eta_i + \eta_j$, 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 $\eta_i + \eta_j \sim \mathcal{N}(0, \sigma_j^2)$. Therefore, equation 5.4 holds according to the analytic Gaussian mechanism (Definition 5).

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 1), the overall privacy guarantee is bounded by $(\max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \delta)$-DP. □

We note that we use the analytic Gaussian mechanism (Definition 5) to calibrate noise in the algorithm. That is, the Gaussian variance is calculated by $\sigma = \alpha\Delta q/\sqrt{2\epsilon}$ where α is a parameter determined by ϵ and δ [2]. This algorithm framework can also be applied to other Gaussian related noise adding mechanisms [54].

5.2.2 Updating Synopses

We first introduce the concept of global and local DP synopses and then discuss the updating process in our additive Gaussian mechanism. A DP synopsis (or synopsis for short) is a noisy answer to a (histogram) view over a database instance.

Global and Local DP Synopses. To solve the maximum query answering problem, for each view $V \in \mathcal{V}$, *DProvSQL* 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, *DProvSQL* also maintains a *local DP synopsis* for each analyst $A_i \in \mathcal{A}$, denoted by $V^{\epsilon'}_{A_i}$, where the local synopsis is always generated from the global synopsis V^{ϵ} of the view V by adding more noise. Hence, we would like to ensure $\epsilon > \epsilon'$. 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 Gaussian Mechanism* (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 for adding noise and ensuring DP. The additive GM calculates σ and σ' 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.

Updating Global Synopses by Combining Views. When the global DP synopsis V^{ϵ} is not sufficiently accurate to handle a local synopsis, *DProvSQL* spends additional privacy budget $\Delta\epsilon$ to update the global DP synopsis to $V^{\epsilon+\Delta\epsilon}$. 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 their budget [43]. That is, for the n-th release,

$$V' = \sum_{i=1}^{n-1} w_i V^{\epsilon_i} + w_n V^{\Delta\epsilon},$$

where $\sum_{i=1}^{n} w_i = 1$ and $v = \sum_{i=1}^{n-1} w_i^2 \sigma_i^2 + w_n^2 \sigma^2$ (whose closed-form solution is $w_i = v/\sigma_i^2$, i.e. $w_i \propto \epsilon_i$).

Lemma 1 (Correctness of the View Combination [43])**.** *The combined DP synopsis in the n-th release satisfies $(\sum_{i=1}^{n-1} \epsilon_i + \Delta\epsilon)$-DP and the expected squared error is less than or equal to v.*

We note that the combination is not *frictionless*. Although the combined synopsis $V^{\epsilon+\Delta\epsilon}$ achieves $(\epsilon + \Delta\epsilon, \delta)$-DP, if we spend the whole privacy budget on generating a synopsis all at once, this one-time view V^* can achieve the same level of privacy but has less expected error (i.e. higher utility) than $V^{\epsilon+\Delta\epsilon}$. We especially remark that the problem of how to design a frictionless updating strategy for Gaussian-related DP mechanisms is interesting and non-trivial on its own right, but out of the scope of this paper.

Updating Local Synopses. When a local DP synopsis $V_{A_i}^{\epsilon'}$ is not sufficiently accurate to handle a query, but the budget ϵ' is still smaller than the budget for the global synopsis, *DProvSQL* generates an intermediate local synopsis $V_{A_i}^{\Delta\epsilon}$ from the global synopsis using additive GM. Then it combines $V_{A_i}^{\Delta\epsilon}$ with the previous local synopsis in a similar way for the global synopses, which leads to a new local synopsis $V_{A_i}^{\epsilon'+\Delta\epsilon}$.

We use the following running example to show the synopses generation and updating process.

Example 3. *To give a concrete example of managing the global and local synopses, we take two data analysts, Alice and Bob, asking the same counting query q_1 (as in Example 2) over database D. We assume the query q_1 can be answered using the view V. We omit the privacy privilege checking for simplicity in this example. When the system is running, Alice first asks the query q_1 with accuracy requirement translated to privacy budget 0.5. We will generate a global synopsis $V^{0.5}$ from the view V with budget 0.5 and then generate a local synopsis $V_{Alice}^{0.5}$ from the global synopsis $V^{0.5}$ for Alice. In this case we can simply copy it, and answer Alice's query. Next, Bob asks the query q_1 with accuracy requirement translated to privacy budget 0.3. Since the budget $0.3 < 0.5$, we can use additiveGM algorithm to generate a local synopsis $V_{Bob}^{0.3}$ from the global synopsis $V^{0.5}$ for Bob and return the query answer based on the local synopsis.*

Assume that Bob realizes q_1 is an important query to him after receiving the noisy answer and he would like to see a more accurate one. Bob asks the query q_1 again with accuracy requirement translated to privacy budget 0.7, which is greater than 0.5, the one associated with the global synopsis $V^{0.5}$. Then we need to update the global synopsis by generating a fresh global synopsis $V^{0.2}$ from the view, and combine $V^{0.5}$ and $V^{0.2}$ to $V^{0.7}$. We also have to update Bob's local synopsis $V^{0.3}_{Bob}$ by generating a fresh local synopsis $V^{0.4}_{Bob}$ from $V^{0.7}$ and then combine $V^{0.3}_{Bob}$ and $V^{0.4}_{Bob}$ to $V^{0.7}_{Bob}$. This updated local synopsis can answer Bob's query.

Remark. Without loss of generality, we can assume the data utility requirements per query requested by the same data analyst never decreases. That is, for the same query, the data analyst is only interested in a more accurate result (with less expected squared error). We remark that in real-world implementation and deployment there are cases where the data analyst can ask a query with decreased utility requirement comparing to historical queries. For example, the data analyst may forget or lose the returned results for historical queries. *DProvSQL* can handle these cases by answering this query based on the corresponding *local* synopsis, sampling a fresh noise and adding to the query result that satisfies the utility requirement. This process does not consume additional privacy budgets, suggested by the post-processing property (formally stated in Proposition 1).

Proposition 1. *Given a query q_i and an expected squared error v_i submitted by data analyst A_i, if v_i is greater than a error v'_i which is associated with the query q_i specified by A_i previously, answering this query using the local synopsis $V^{\epsilon_i}_{A_i}$ does not consume additional privacy budget.*

5.2.3 Accuracy-Privacy Translation

Differing from the accuracy-privacy translation in the baseline solution, the additive Gaussian approach maintains the global DP synopses for answering queries. This leads to the research question for *DProvSQL*: how to translate the utility requirements into privacy budgets with the existence of the release of *historical queries*? To be more specific, the goal of the privacy translation in the additive Gaussian approach is to find the minimum privacy budget to update the global synopsis such that the updated synopsis can be used to answer the incoming query constraint to the query utility requirement. As we show in Section 5.2.2, the synopses updating and combining is not *frictionless* (i.e., lossless) comparing to simply adding the privacy budgets to the desired guarantee. To solve this issue, we propose the accuracy-privacy translation paradigm for the additive Gaussian approach.

This translation paradigm works for queries where their global sensitivity is bounded and can be calculated independent of the underlying database instance. This type of queries includes linear counting queries, histogram queries and so on. As shown in Algorithm 4: 13, this accuracy-privacy translator module takes the query q_i, the utility requirement v_i, the view 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).

The first observation is that the first query release for the view does not involve the frictional updating issue. Then we separate the translation into two phases where the first query release directly follows the analytic Gaussian translation in our baseline approach.

As for the second phase translation, recall that if we have a global DP synopsis V^{ϵ_i} at hand (with expected error $v_i = \sigma_i^2 \propto 1/\epsilon_i^2$) for a specific query-view and a new query is submitted by a data analyst with expected error $v_i' < v_i$. The translation cannot solely find a minimum ϵ_i' according to the global sensitivity and the accuracy requirement through the above translation approaches. This is because when combining the views V^{ϵ_i} and $V^{\Delta\epsilon}$ where $\Delta\epsilon = \epsilon_i' - \epsilon_i$, the expected error for the resulting combined view can be larger than the error specified by data analyst. Thus, our privacy translation module should take this accuracy loss into account. To achieve this, we include the optimization problem into the privacy translation. That is, we first calculate the Gaussian variance of the current DP synopsis σ_c and solve the following problem given the accuracy requirement v_i'.

$$\begin{aligned} \underset{\sigma}{\text{maximize}} \quad & v = w_1^2\sigma_c^2 + w_2^2\sigma^2 && (5.5)\\ \text{subject to} \quad & w_1 + w_2 = 1 && (5.6)\\ & w_1, w_2 \in [0,1] && (5.7)\\ & v \leq v_i' && (5.8) \end{aligned}$$

By solving this optimization algorithm, we can obtain the minimal error variance σ^2. By translating this into privacy budget using the standard analytic Gaussian translation technique (Definition 14), we can get the minimum privacy budget that achieves the required accuracy guarantee.

Theorem 6. *Given a query q where its global sensitivity is bounded and the historical released DP synopsis V^ϵ for answering q, the additive Gaussian approach (Algorithm 4) returns the query result with the expected squared error at most v and satisfies differential privacy with a minimal cost of privacyTranslate$(q, v, V, V^\epsilon).\epsilon$.*

5.2.4 Provenance Sanity Checking

The provenance sanity checking for additive Gaussian approach is similar to the counterpart for the baseline approach. We would like to highlight two differences between the two methods. First, thanks to the additive Gaussian mechanism, the composition across analysts is bounded as tight as $\max \epsilon_i$. Therefore, the column-level composition is substituted with the MAX function. Second, since we update the privacy budget recorded in the privacy provenance table to ϵ_i, we need to subtract the historical budget in that entry when checking the constraints.

5.3 Privacy Guarantee

The overall system privacy guarantee is given by the following theorem.

Theorem 7. *Given the privacy provenance table and its constraint specifications,* $\Psi = \{\psi_{A_i} | A_i \in \mathcal{A}\} \cup \{\psi_{V_j} | V_j \in \mathcal{V}\} \cup \{\psi_P\}$*, Algorithm 1 ensures* $[\ldots, (A_i, \psi_{A_i}, \delta), \ldots]$*-multi-analyst-DP; it also ensures* ψ_{V_j}*-DP for view* $V_j \in \mathcal{V}$ *and overall* ψ_P*-DP if all the data analysts collude.*

As a proof sketch, this theorem will hold due to the enforcement of the privacy provenance table and the correctness of the composition over the privacy provenance table, assuming the privacy guarantee given by the additive Gaussian noise adding mechanism (as proved in Theorem 5).

Algorithm 4: Additive Gaussian Approach

Set δ in the system
Function *run(* $P, A_i, V_{A_i}, \epsilon_i$ *)* **:**
  if $\epsilon_i > \epsilon$ *for global synopsis* V^{ϵ} **then**
    $\Delta\epsilon = \epsilon_i - \epsilon$
    $V^{\Delta\epsilon} \leftarrow$ ADDITIVEGM$(\{A_i, A_{j,V^{\epsilon}=V^{\epsilon}_{A_j}}\}, \{\Delta\epsilon, \epsilon_i\}, q, D)$
    Update global synopsis to $V^{\epsilon \leftarrow \epsilon_i}$ with $V^{\Delta\epsilon}$
  $\Delta\epsilon = \epsilon - V_{A_i}.\epsilon$
  $V^{\Delta\epsilon}_{A_i} \leftarrow$ ADDITIVEGM$(\{A_i, A_{j,V^{\epsilon}=V^{\epsilon}_{A_j}}\}, \{\Delta\epsilon, \epsilon\}, q, D)$
  Update local synopsis to $V^{\epsilon' \leftarrow \epsilon_i}_{A_i}$ with $V^{\Delta\epsilon}_{A_i}$
  Update privacy provenance table $P[A_i, V] \leftarrow \epsilon_i$
  return $r_i \leftarrow V^{\epsilon'}_{A_i}$
end
Function *privacyTranslate(* q, v, V, V^{ϵ} *)* **:**
  $v' \leftarrow$ ESTIMATEERROR(q, v, V^{ϵ})
  $w \leftarrow$ MINIMIZER(func=$-\frac{(v-w^2v')}{(1-w)^2}$, bounds=$[0, 1]$)
  $v_t \leftarrow \frac{(v-w^2v')}{(1-w)^2}$ ▷ Target error bound w/ friction.
  $\epsilon \leftarrow$ PRIVACYTRANSLATE(q, v_t, V) ▷ Baseline translation.
  return ϵ
end
Function *constraintCheck(* $P, A_i, V_j, \epsilon_i, \Psi$ *)* **:**
  status = True
  $\epsilon' = P[A_i, V]$
  if *P.composite(axis=Row, r=P.max)* $+ \epsilon_i$ - ϵ'¿$\Psi.\psi_P$ **then**
    return False
  if *P.composite(axis=Row, r=V)* $+ \epsilon_i - \epsilon' > \Psi.\psi_{A_i}$ **then**
    return False
  if *P.max(axis=Column, c=*A_i*)* $+ \epsilon_i - \epsilon' > \Psi.\psi_{V_j}$ **then**
    return False
  return status
end

Chapter 6

Implementation and Evaluation

In this chapter, we describe the system implementation details and the results for empirical evaluation.

6.1 System Implementation

We implement *DProvSQL* in Scala 2.12.2 and use sbt as the system dependency management tool. We use PostgreSQL as the underlying database platform and utilize the Breeze library for noise calibration and solving the optimization problem in privacy translation (in the additive Gaussian approach). *DProvSQL* works as a middleware between the data analysts and the existing DP SQL query system, and provides advanced functionalities such as privacy provenance, query control, query answering over the cached views, and privacy translation. *DProvSQL* 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 *DProvSQL* over Chorus [25], which is an open-source DP query answering system. That is, *DProvSQL* 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). Since Chorus only supports Laplace-related DP mechanisms, we implement the analytic Gaussian mechanism [2] in Scala and incorporate it into the system. We build our privacy provenance table and implement a query transformation module that can transform the incoming queries into the linear queries over the DP views/synopses.

6.2 Empirical Study

In the empirical evaluation, we would like to evaluate the efficacy and the efficiency of *DProvSQL* by comparing with baseline systems. We test the experiments on a machine with MacOS system and with the following hardware configuration: Apple M1 chip, with 8GB memory size. The goal of the experiments is to show that in the use cases with the presence of multiple data analysts, *DProvSQL* can answer much more queries (i.e., better system utility) than baseline or existing systems, while we would like to analyze from which modules or design choices we can get such benefits.

6.2.1 Experiment Setup

Datasets. We conduct the experiments on 2 different datasets, the Adult dataset [12] and the TPC-H dataset [7]. The Adult dataset describes the census demographic data, which includes 15 attributes and has 45,224 rows. The TPC-H dataset we use is a synthetic dataset that joins 8 relations (supplier, part, customer, partsupp, orders, lineiterm, nation, and region), describing the customer-sales relationships and the company's sales records. We use the TPC-H generator to generate the dataset of 1 GB data and import them into PostgreSQL.

Metrics. We use three metrics to evaluate our system: a) Utility. Utility measures the number of queries that can be answered by the system when the overall privacy budget exhausts. b) Performance. We measure the query processing run time in milliseconds. c) Fairness. We additionally measure the fairness by using the DCFG metric we propose in Definition 11.

Baselines. We consider three baselines as compare to our additive Gaussian approach. The first one is the plain Chorus mechanism, where we only set up the overall privacy budget in the system. The second one is the Chorus mechanism with the privacy provenance table, where we would like to see the benefit from having the privacy provenance table. The last one is to equip Chorus with our baseline approach. That is to enable query answering with views while enforcing individual constraints over the privacy provenance table.

Use Cases. We consider the case where we have three data analysts. Two of them has low privacy privilege and the other one has higher privacy privilege. When the overall privacy budget is set to 2, we set low privilege and high privilege as 0.5 and 1, respectively. The individual privacy privilege limit scales up proportional to the overall privacy budget when it is set to 4, 6, and 8 in our experiments. We consider to build a single view for

Table 6.1: The comparison between our approach and baseline approach (in terms of the number of queries being answered, and the minimum expected error of answers, denoted by v).

	Analyst 1	Analyst 2	Analyst 3
Chorus	2 (v=39)	2 (v=39)	5 (v=36)
Our Approach	15 (v=26)	15 (v=26)	26 (v=15)

experiments on both dataset. On Adult dataset, we build a 3-way contingency table over attributes age, gender, and education as a view. On TPC-H dataset, we build a 2-way histogram over the p_brand and p_size attributes.

We design two ways of generating incoming queries: a) **Take-turns.** The data analysts take turns to ask queries. If the same query is asked by the data analyst for a second time, the accuracy requirement never decreases, which simulates the real-world situations where the data analysts would like to see a more accurate results of this query. b) **Random.** Given a set of queries, a set of accuracy requirements, and the set of data analysts, we randomly choose a data analyst to ask a query with an accuracy requirement every time.

6.2.2 Empirical Results

We report the empirical results we obtain on the two datasets.

Empirical results on Adult dataset. We only consider the case where among the three data analysts, two of which has low privilege level (row constraints $\phi_{A_1} = \phi_{A_2}$=1) and the other one with high privilege level (ϕ_{A_3}=2). For simplicity, we assume all data analysts ask the same query:

```
SELECT * FROM adult
WHERE age >= 39 AND education = 'Bachelors';
```

The data analysts keep on submitting this query with higher accuracy requirement (specified by the expected squared error, starting from 40, each time decreasing by 1) over time. We enable the row constraints over different analysts while the column constraint is naturally enforced since we only have a single view in the preliminary experiments. We compare our approach to Chorus mechanism, which regard each query from analysts as a separated query and answer it independently. We measure the number of queries that the system could support for each data analyst until no more queries can be handled without

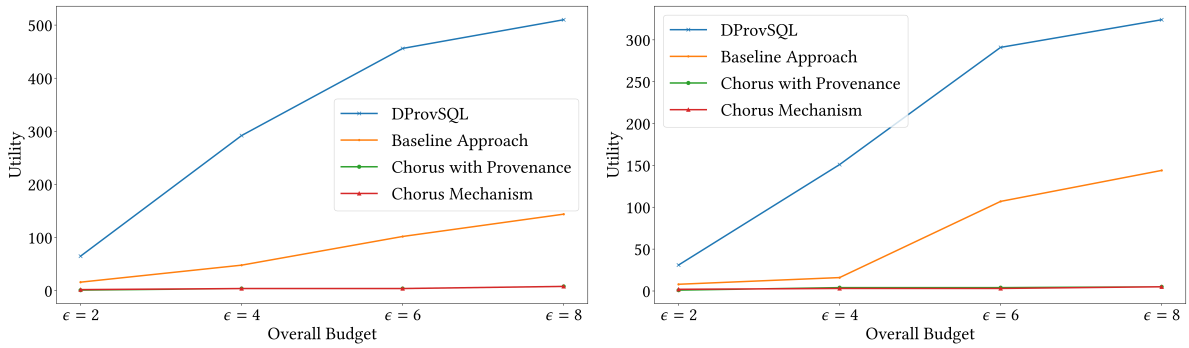

Figure 6.1: Utility v.s. the overall budget of the system: a) Take-turns; b) Random.

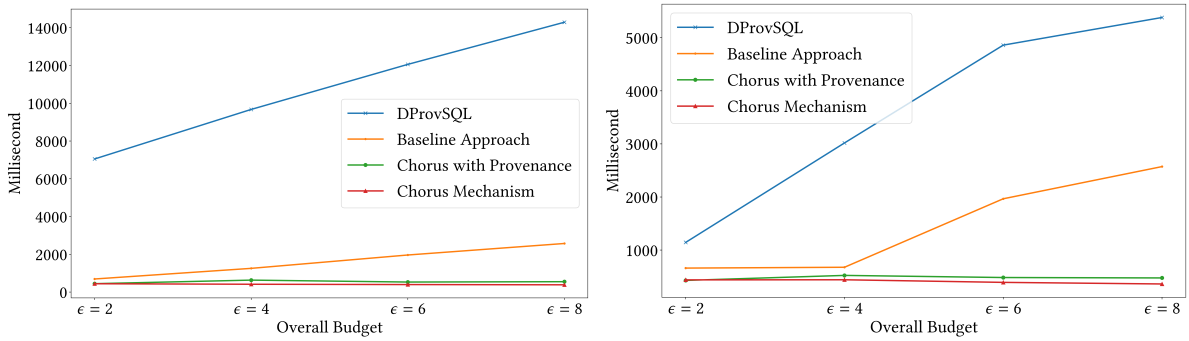

Figure 6.2: Performance v.s. the overall budget of the system: a) Take-turns; b) Random.

violating the privacy constraints, and the minimum expected error among all queries returned to each data analyst. As shown in Table 6.1, our approach can answer 6.2x more queries than the baseline on average meanwhile the answer from our approach is 1.5x to 2.4x more accurate. Our approach performs significantly better than the baseline, because in the baseline, every query-answering is independent, whereas the usage and management of global/local DP synopses in our approach enables correlated DP noise to the query results. Our mechanism can therefore avoid wasting budget and hence answer more queries accurately.

Empirical results on TPC-H dataset. We perform more detailed experiments on TPC-H dataset. In particular, we consider 4 different queries that can be answered over the 2-way histogram view. For example, one of the queries is the following.

```
SELECT count(p_brand)
FROM part
WHERE p_size < '30' AND p_brand = 'Brand#14'
```

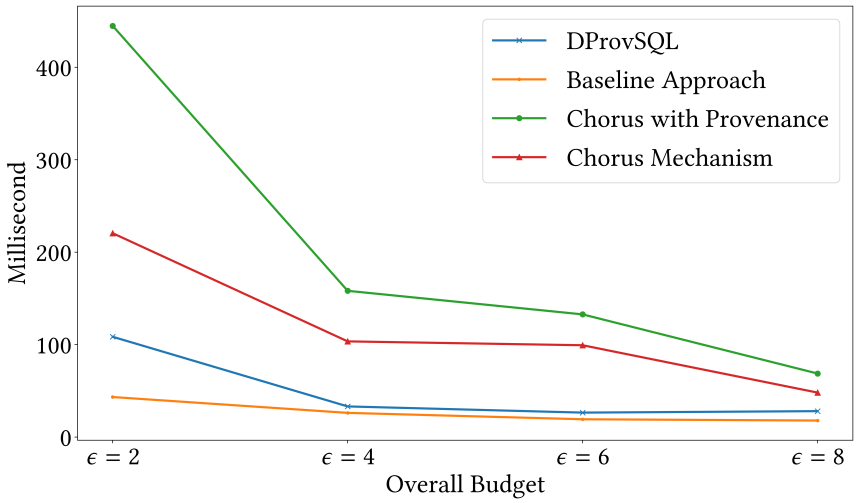

Figure 6.3: Per query performance v.s. the overall budget of the system: Take-turns.

The comparison among 4 mechanisms in terms of utility is shown in Figure 6.1. It is clear that our *DProvSQL* with the additive Gaussian approach achieves the best utility, with 4x to 5x improvement over the baseline approach in the "Take-turns" use case, but the same trend can be also observed in the "Random" use case. Furthermore, since the baseline approach adopts the view based query answering paradigm, we can see that it works much better than naïvely adopting Chorus or simply plugging in the privacy provenance table into Chorus.

Figure 6.2 shows the comparison of system performance among the mechanisms. The introduction of the privacy provenance table and the constraint checking to the system introduces overhead, observed from the performance of Chorus and Chorus with provenance. At first glance, *DProvSQL* approach has large system performance overhead comparing to the other baselines. However, this experiments shows the overall query processing overhead. A closer look at the per query performance (Figure 6.3) suggests that *DProvSQL* and baseline approach has lower per query overhead than the Chorus and Chorus with provenance approach. This is because if queries can be directly answered using an existing synopsis, then we will not spend time to do the privacy translation and the provenance checking. *DProvSQL* has marginal overhead over the baseline approach, which is reasonable, because the additive Gaussian approach introduces additional optimization problem to solve.

Lastly, we briefly discuss the fairness measurement in our system. Figure 6.4 shows the fairness scores of each mechanism. Since the DCFG is a metric which is not normalized,

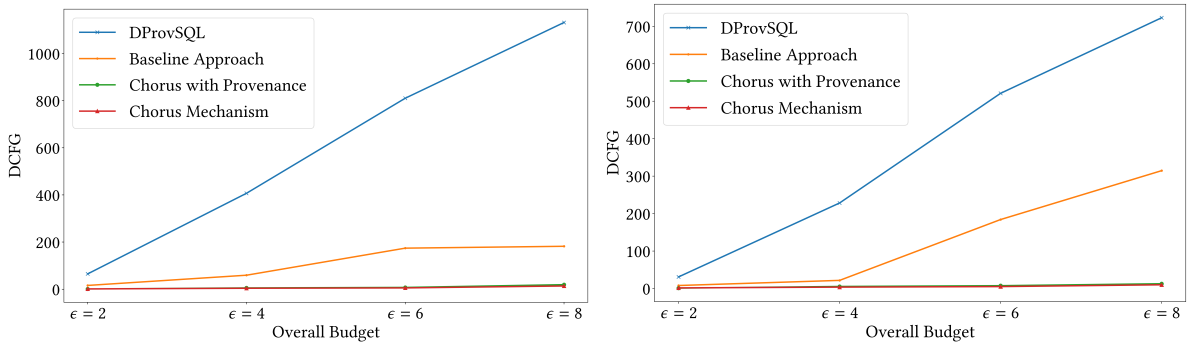

Figure 6.4: Fairness (DCFG) v.s. the overall budget of the system: a) Take-turns; b) Random.

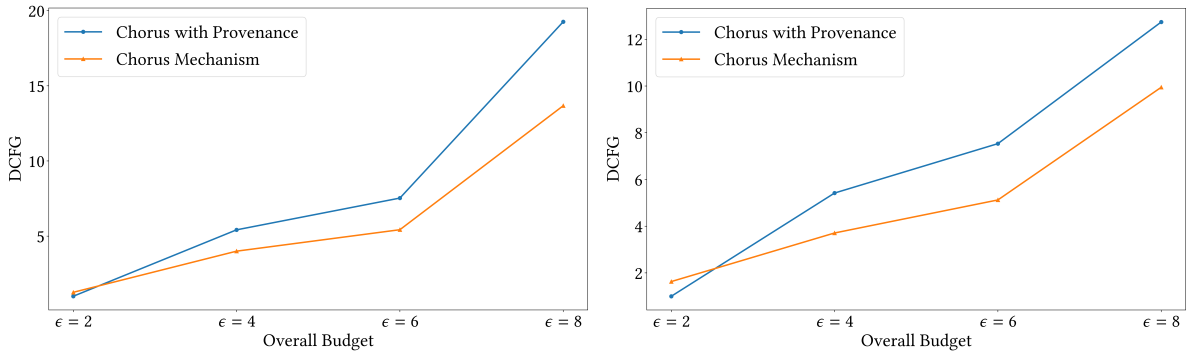

Figure 6.5: Fairness comparison between Chorus mechanism and Chorus with the privacy provenance table: a) Take-turns; b) Random

the mechanism that answers more queries naturally wins more scores on it. Thus, we look into the comparison between the Chorus mechanism and the Chorus with provenance mechanism, where those two answer the similar number of queries, if not the same. This comparison is plotted in Figure 6.5. One can observe that, after adopting the privacy provenance table, the fairness scoring gets increased with Chorus. This result is intuitive, because the privacy provenance table can prevent data analysts with lower privilege level asking too many queries so that the analysts with higher privilege can hardly submit queries before the overall budget exhausts.

Chapter 7

Related Work

We discuss related works in this chapter. Such related works include prior effort on DP systems [36, 44, 26, 30, 39, 51, 1, 18] or programming frameworks [25, 34], variants of DP frameworks [27, 16, 29], and the theoretical wisdom on providing tighter instance-based bound on answering join queries with DP [11, 10, 9]. We describe and discuss these prior works in details below.

7.1 Existing DP Query Systems

PINQ [36] and wPINQ [44] PINQ [36] may be the first-of-its-kind end-to-end interactive data query and analysis system that enforces differential privacy. PINQ is implemented with the C#'s LINQ language and allows the users to submit queries through a SQL-like query interface. PINQ analyzes the bound of privacy loss in terms of tracking the *stability* of the data transformation according to the query answering. Therefore, PINQ can support a class of database transformation queries including SELECTION, PROJECTION, COUNT, and COUNT DISTINCT. PINQ implements a special form of GROUP BY query in which the querier must specify a list of grouping keys/elements. PINQ supports a limited class of join queries where the data is first grouped by the join keys and then the groups are joined by the group keys. This restriction of join queries is enforced to bound the query sensitivity, because the global sensitivity of the standard JOIN operator is unbounded. wPINQ [44] extends PINQ in terms of associating weights to each row of the table and trimming the weights in the JOIN operator to obtain a sensitivity of 1. Thus, wPINQ can support a wider class of equi-join queries than PINQ.

FLEX [26] Johnson et al. propose FLEX [26], a DP system that can support a large class of general equi-join queries. FLEX adopts the natural extension of the standard DP definition in the multi-relation database where two neighbouring databases only differ in one row in a single relation. Since the global sensitivity of a equi-join query is unbounded, FLEX proposes *elastic sensitivity*, which is a instance-dependent upper bound on the local sensitivity [41] but computationally much more efficient than calculating local sensitivity. FLEX develops rules to calculate stability and maximum frequency of a (key) attribute and therefore bound the elastic sensitivity. By adding Laplace noise proportional to the smooth upper bound on elastic sensitivity, they build FLEX mechanism and the system to achieve (ϵ, δ)-DP guarantee.

Chorus [25] Chorus [25] is a programming framework built upon FLEX [26] and implemented as a Scala library, which provides principled and scalable database querying with differential privacy guarantee. Chorus consists of a three-module structure, i.e., the rewriting-analysis-postprocessing structure – it takes a query as input from the data analyst, rewrites the query if a clipping bound should be applied, analyzes the sensitivity of the query, and finally injects proper noise or combines the results. Chorus is designed to be independent of the underlying database and supports a broad spectrum of DP mechanism that users from DP novice to domain expert can make use of this framework. *Both FLEX and Chorus systems only track a sole global privacy budget which will be subtracted when each incoming query is answered.*

PrivateSQL [30] Differing from FLEX, PrivateSQL [30] introduces a new definition of differential privacy for multiple relations. The new privacy notion captures the foreign key constraints among different relation schemas, specified by privacy policies [23], and can therefore express richer privacy semantics such as Edge-DP and Node-DP. PrivateSQL answers queries based views and differentially private synopses. PrivateSQL calculates the (truncated) global sensitivity for view based on rules similar to those of FLEX. The sensitivity is truncated for join queries in order to bound the influence of the removal of a tuple to other relations, and the truncation bound is privately learnt using the sparse vector technique. *PrivateSQL has a privacy budget allocator that splits a total global budget to each view, but this allocation is static and only depends on some heuristics such as fair splitting.*

7.2 Existing Work on Highly Sensitive Queries

Residual sensitivity [11, 10] Residual sensitivity, proposed by Dong and Yi [11, 10], is yet another instance-dependent upper bound on local sensitivity for answering join queries. The residual sensitivity [11] is developed upon the residual query of a multi-way join query and the maximum boundary of the residual query on a database instance. The maximum value of the maximum boundary of residual queries with arbitrary changes made in the private relations is proved to yield an upper bound of local sensitivity. Taking advantage of this property, residual sensitivity provides a tighter upper bound on local sensitivity of multi-way join queries than elastic sensitivity [26], and can be efficiently calculated (in polynomial time). Their concurrent work [10] complements the results by proving the instance-optimality on specific complex queries (i.e., the conjunctive queries).

R2T [9] Mechanisms adding noise proportional to the smooth sensitivity may lose the foreign key constraint when handling join queries. Another line of research is based on truncation, that is, to bound the user contribution by truncating the join table. The truncation operation introduce bias in the query answer but can reduce the noise variance, which has been shown to be useful in answering aggregation queries [25] and join queries with foreign key constraints [30]. The privacy guarantee is proved via leveraging the Lipschitz extension of DP [47]. A recent work, Race-to-the-Top (R2T) [9], shows a systematic approach in using truncation to answering general Select-Projection-Join-Aggregation (SPJA) queries with instance-optimality. R2T adaptively choose the truncation factor in combination with certain DP truncation mechanisms to achieve the instance-optimal error. By extending a LP-based mechanism, R2T can generalize to answering any SPJA query.

7.3 Other Related DP Frameworks

Personalized differential privacy [27, 16] Jorgensen et al. [27] propose personalized differential privacy (PDP) as a variant of the DP framework. In constrast to the standard DP, PDP considers the case that *users contributing to the protected database can have different privacy levels or requirements*. [27] shows PDP can be achieved by introducing an additional tuple-level sampling based modification to the traditional DP mechanisms. PDP avoids neither insufficient protection for some users nor over-protection for others, and therefore can gain benefits of improving data utility for certain scenarios. Concurrently, Ebadi et al. [16] propose yet another PDP framework from the formal methods perspective that serves a similar purpose. In their framework, a privacy provenance approach is presented to track the privacy loss of each *individual record in the database*. This

provenance module is implemented and incorporated into PINQ. *Both PDP frameworks from [27] and [16] can be regarded as dual frameworks to ours, in the sense that they would like to provide fine-grained privacy tracking for users in the dataset while we are bounding the privacy loss on the multiple data analysts side.*

One-sided differential privacy [29] Kotsogiannis et al. propose one-sided differential privacy (OSDP) [29] as a DP variant that handles the case where the protected database can be partitioned into sensitive data and non-sensitive data. Regarding all data as equally sensitive, standard DP can be over-pessimistic to provide a less useful solution. OSDP is such a framework that intends to leverage the non-sensitive data to improve the data utility (or query accuracy) while not releasing the whole non-sensitive data since this leads to the inference of the sensitive data. OSDP is demonstrated to be used to answer counting queries and publishing datasets with complex data type.

DP systems with the support of accuracy specification (APEx [18] and DPella [34]) Most DP systems require data analysts to submit a privacy budget (ϵ) associated with the query that indicates the share of budgets they would like to spend on this query. However, the budget as a privacy indicator may not be intuitive for the non-DP-expert data analysts to obtain the analysis results they desired. Some pioneer existing work such as APEx [18] and DPella [34] start to investigate an accuracy-aware DP system for data analysis and exploration. In such systems, the data analysts are allowed to submit the accuracy requirement instead of the privacy budget with their queries. The systems utilize an accuracy translation module that can convert the accuracy requirement into an estimated privacy budget interval automatically. The systems answer the query and tell the privacy cost to the analysts if the total budget is not exceeded; otherwise the query will be rejected. However, such systems are limited in terms of the supported query types. For example, APEx only supports different counting queries such as workload queries and iceberg queries.

DP framework involving multiple data analysts [53, 28, 45] Very recently, some research works start to investigating DP frameworks that involve multiple data analysts as a more realistic setting. Pujol et al. [45] study the problem of releasing data to multi-analyst with a limited shared privacy budget, where the data release should satisfy the desiderata of sharing incentive, non-interference, and adaptivity, and they design mechanisms to achieve these goals. Other works, e.g. fitness-for-use [53], consider the per analyst accuracy requirements and propose mechanisms to optimize the injected noise to meet the accuracy requirement as per data analyst and minimize the total privacy loss. None of the existing works so far consider the same problem as ours.

Chapter 8

Discussion and Future Work

In this chapter, we discuss the limitations of *DProvSQL* (in terms of the multi-analyst DP framework and the proposed system) and the potential directions for future work.

8.1 Preliminary Work on Highly Sensitive Queries

The algorithms or mechanisms described in Section 5 can only be used to answer queries where their global sensitivity can be bounded. For queries with the join operator over a multi-relational database, the global sensitivity of these queries is, unfortunately, unbounded. This is due to the definition of global sensitivity, where, through the join operator, adding or removing one row in a table can affect an arbitrary number of rows in another table. Therefore, existing works propose instance-dependent sensitivity definitions, including local sensitivity [41], elastic sensitivity [26], and residual sensitivity [11], and inject noise proportional to the smooth upper-bound of the sensitivities [41] to achieve differential privacy.

In this section, we design specific algorithms to handle high-sensitivity queries which involve the join operators. First, we naturally generalize the Multi-analyst DP definition to the multi-relational database setting. Similar to DP, the generalization of Multi-analyst DP mainly results in the generalization of the definition of neighbouring database instances. Given a database instance D with relations $\mathcal{R} = \{R_1, \ldots, R_l\}$, a subset of the relations $\mathcal{R}_{Priv} \subseteq \mathcal{R}$ are private and the remaining relations $\mathcal{R}_{Pub} \coloneqq \mathcal{R} \backslash \mathcal{R}_{Priv}$ are public. Following the generalization of DP in multi-relational DB settings [26, 11], we consider the two database instances that are *neighbouring databases* if they only differ in 1 tuple (i.e. adding

or removing 1 tuple as we consider unbounded DP) in one of the private relations $R \in \mathcal{R}_{Priv}$. Then the local sensitivity of a query q evaluated at a database instance D is defined as

$$LS_q(D) = \max_{D', d(D,D')=1} ||q(D)| - |q(D')||.$$

Note that the global sensitivity of query q is the maximum local sensitivity, i.e., $GS_q = \max_D LS_q(D)$.

The first challenge behind translating the accuracy requirement into a privacy budget for join queries over a multi-relational database is that the instance-dependent sensitivity may leak information about the underlying database instance. Thus, we need to privately bound the global sensitivity of the instance-dependent local sensitivity.

We start to study the problem by considering the following two-way join-counting query on the attribute T of private relations $r1$ and $r2$.

```
SELECT COUNT(*)
FROM r1 JOIN r2
ON r1.T = r2.T;
```

We denote the frequency of the element which has the maximum occurrence of an attribute A in relation r in the database instance D by $mf(T, r, D)$. In addition, for an element $e_i \in T$, let $f(e_i, r, D)$ be the frequency of the occurrence of e_i in relation r and database instance D. Then we show the global sensitivity of the local sensitivity of the join-counting query by proving the following lemma.

Lemma 2. *The global sensitivity of local sensitivity of the join-counting query is bounded by 1.*

Proof. The global sensitivity of local sensitivity of the query can be written as

$$GS_{LS_q} = \max_{D, D', d(D,D')=1} |LS_q(D) - LS_q(D')| \tag{8.1}$$

$$= \max_{D, D', d(D,D')=1} \left| \max_{D_1, d(D,D_1)=1} |q(D) - q(D_1)| - \max_{D_1', d(D_1', D')} |q(D') - q(D_1')| \right| \tag{8.2}$$

We first discuss the two-way join query (on attribute T of private relations r_1 and r_2) and then generalize to multi-way joins. If we consider unbounded DP, there are two

possible cases transforming from the database instance D to D': 1) add (or remove) 1 row in r_1, 2) add (or remove) 1 row in r_2. Without loss of generality, we can assume $mf(T, r_1, D) > mf(T, r_2, D)$, or equivalently, $mf(T, r_1, D) \geq mf(T, r_2, D) + 1$. Then we analyze both cases.

Case 1. If we add (or remove) 1 row in r_1, to maximize the global sensitivity, the added (or removed) row should contain the element e_i in column T, in which $e_i = \max_i\{e_i \in T \mid f(e_i, r_1, D) = mf(T, r_1, D)\}$. Then in database D', $f(e_i, r_1, D') = f(e_i, r_1, D) \pm 1$ (+1 for adding a row and -1 for removing a row). In the case that D' is obtained by adding the row to D, the local sensitivity of query q on database instance D' can be calculated as $LS_q(D') = f(e_i, r_1, D')$ which is achieved by adding/removing a row containing e_i in relation r_2. In the case that we remove the row to get D', $LS_q(D') = \max\{f(e_i, r_1, D) - 1, mf(T, r_2, D)\} = f(e_i, r_1, D) - 1$. Looking at the database instance D, the local sensitivity $LS_q(D) = mf(T, r_1, D) = f(e_i, r_1, D)$ (considering adding/removing a row in r_2). In both cases of transforming D to D', the global sensitivity of the local sensitivity is bounded by $GS_{LS_q} = 1$.

Case 2. Similarly, if we add (or remove) 1 row in r_2, the added (or removed) row should contain the element e_j such that $e_j = \max_j\{e_j \in T \mid f(e_j, r_2, D) = mf(T, r_2, D)\}$. Then in database D', we have $f(e_j, r_2, D') = f(e_j, r_2, D) \pm 1$. If we add a row to get D', $LS_q(D') = \max\{f(e_i, r_1, D), f(e_j, r_2, D')\}$, considering adding or removing a row containing e_i in r_2 or e_j in r_1. This equation can be simplified to be $LS_q(D') = \max\{f(e_i, r_1, D), f(e_j, r_2, D')\} = \max\{mf(T, r_1, D), mf(T, r_2, D) + 1\} = mf(T, r_1, D)$. If we remove a row to get D', $LS_q(D') = mf(T, r_1, D)$. As for the database instance D, the local sensitivity $LS_q(D) = mf(T, r_1, D) = f(e_i, r_1, D)$ (considering adding/removing a row in r_2), which is the same as in Case 1. Since $LS_q(D') = LS_q(D)$ no matter adding or removing a row from D to obtain D', the global sensitivity of the local sensitivity for this case is $GS_{LS_q} = 0$.

Combining two case analyses, the global sensitivity of the local sensitivity of the two-way join-counting query is bounded by 1. This result can be naturally generalized to multi-way join-counting queries. If we enrich the maximum frequency notation to allow it to take the set of join-key attribute instead of the single attribute T, the similar analysis result still holds. □

Remark. Lemma 2 does not only apply to join-counting queries but can also be extended to self-join queries such as the triangle-counting query in a graph, where the database table records the edges (in the way that a row contains the source and destination of an edge). We consider *edge-DP* as the privacy notion, where two neighbouring graph databases differ in one edge. The global sensitivity of local sensitivity is still bounded by 1. The intuition

behind this is that the local sensitivity of this self-join query is equivalent to the maximum degree of the vertices in this graph. Therefore, for any two graphs G and G' that differ in only 1 edge, the maximum degree can only differ in 1. In the future extension of this work, we will develop rules to cover any general join queries.

Definition 15 (Local Sensitivity at Distance k [41])**.** *The local sensitivity of a query q at Hamming distance k, evaluated on a database instance D, which is denoted as $LS_q^{(k)}(D)$, is defined as*

$$LS_q^{(k)}(D) = \max_{D', d(D,D')\leq k} LS_q(D'),$$

where $d(D, D')$ represents the Hamming distance between two databases, i.e., the number of tuples differ in the two instances.

If we allow dummy records in database, the local sensitivity at distance k is equivalent to $LS_q^{(k)}(D) = \max_{D', d(D,D')\leq k} LS_q(D') = \max_{D', d(D,D')=k} LS_q(D')$ [11]. That is, for $\forall k \geq 0$, $LS_q^{(k)}(D) \leq LS_q^{(k+1)}(D)$, where a special case $LS_q^{(0)}(D) = LS_q(D)$. Then we prove the following theorem, showing the global sensitivity of local sensitivity of the specific query is bounded at any distance.

Theorem 8. *The global sensitivity of local sensitivity of the join-counting query at Hamming distance k is bounded by 1.*

Proof. This theorem can be proved by induction on the distance k. **Base case**: as shown in Lemma 2, the global sensitivity of local sensitivity of the join-counting query at distance 0 is bounded by 1. **Induction step**: suppose the global sensitivity of local sensitivity of the join-counting query is bounded by 1 at distance n (i.e. $GS_{LS_q^{(n)}} \leq 1$), we then consider the case that the distance $d(D, D') = n + 1$. To build the connection between the case $k = n$ and $k = n + 1$, we need to prove the *greedy* property of the sensitivity results when increasing the distance between two databases. To show this, we prove the following claim.

Claim 1. $LS_q^{(n+1)}(D) \leq LS_q^{(n)}(D) + 1$

Proof of Claim 1. Given the same database D, at the base case where $n = 0$, as shown in the proof of Lemma 2, $LS_q^{(0)}(D) = mf(T, r_1, D)$. Suppose for the element $e_i \in T$, we have $f(e_i, r_1, D) = mf(T, r_1, D)$. Then consider $LS_q^{(1)}(D)$, which is the maximum difference in the query results over the database instances with distance 2. Compared to $LS_q^{(0)}(D)$, adding or removing 2 rows instead of 1 row in the database instance can cause the result to change at most by 1, which is achieved if we first add or remove a row to get $LS_q^{(0)}(D)$ and then add or remove a row which contains the same element in T to get

$LS_q^{(1)}(D)$. Thus, we have $LS_q^{(1)}(D) \leq LS_q^{(0)}(D)+1$. Extending to case n, $LS_q^{(n)}(D)$ indicates a database instance D' which is at distance n to D. To calculate the local sensitivity $LS_q^{(n+1)}(D)$, the optimal way is to consider the database instance D'' by adding or removing a row that contains the element $e_i = \max_i\{e_i \in T \mid f(e_i, r_j, D') = mf(T, r_j, D'), \forall r_j \in R\}$. Therefore we have $LS_q^{(n+1)}(D) \leq LS_q^{(n)}(D) + 1$.

Given the result of Claim 1, for database D and D', the claim holds simultaneously. Therefore, if at distance n, $GS_{LS_q^{(n)}} \leq 1$, then at distance $n+1$, $GS_{LS_q^{(n+1)}} = \max |LS_q^{(n+1)}(D) - LS_q^{(n+1)}(D')| \leq 1$. This proves the theorem. □

While the details of correctly translating accuracy requirement into privacy budgets for high sensitive join queries are still under development, the intuitive ideas are to use the theorems proved above to privately bound the local sensitivity. Since the global sensitivity of the local sensitivity is bounded and small, we can spare some privacy budgets and use the report noisy max [14] mechanism to get the noisy maximum local sensitivity with distance k. The noise calibration mechanism for answering the join queries can follow the following mechanism [41]. We will describe more details in the later version of this work.

Lemma 3 (Gaussian Noise Calibration [41])**.** *Let $SS(q, D)$ be the β-smooth upper bound on the local sensitivity of a d-dimensional query q on database instance D, where $\beta = \frac{\epsilon}{4(d+\ln(2/\delta))}$. The following noise-adding mechanism satisfies (ϵ, δ)-DP:*

$$\mathcal{M}(D) = q(D) + Gaussian(\frac{5\sqrt{2\ln(2/\delta)} \cdot SS(q, D)}{\epsilon}).$$

8.2 Future Directions

We discuss some limitations of this work and the potential directions for future work. The **first** limitation of this work results in the multi-analysts DP framework, where we consider multiple data analysts with different privacy privilege levels in the query answering system and we would like to bound the overall privacy loss if the data analysts **all collude**. We build the additive Gaussian mechanism and propose our system to maximize the number of queries being answered under this setting. However, this setting is rigid and can be relaxed to be more realistic by introducing a colluding parameter t among data analysts, similar to the multi-party computation threat model [13]. That is, while all data analysts may not collude, a part or group of data analysts can. Accordingly, this parameter t represents

the colluding portion of data analysts; the model we use in this paper is subsumed by this t-collusion notion since when $t = 1$, it degrades to our setting. How to make use of this property, design algorithms, and analyze the effect of privacy loss still remain active research questions and could be considered for future work.

Second, we have described how to extend our algorithms and system to answer high-sensitive queries such as join queries over a multi-relational database. However, we only have proved that the proposed methods work for a specific group of join queries by privately bounding the global sensitivity of the local sensitivity of such queries. This method does not naturally generalize to any upper bound of the local sensitivity (like elastic sensitivity [26] or residual sensitivity [11]). In future work, we will develop rules to bound the global sensitivity of the upper bound on local sensitivity and extend the applicable queries to any join queries. Furthermore, other than adding noise to the smooth upper bound of the local sensitivity, the truncation-based methods [9] are also useful in answering high-sensitive differentially private queries. How to translate the accuracy requirements into a privacy budget with truncation-based methods and fits such a module into our system framework is another interesting research question for future work.

Third, the system performance can be optimized by a more careful design of the view and synopses and by introducing a fine-grained way of updating the synopses. Making use of the sparsity in the data itself [52] or using data-dependent views [32] can result in adding less noise or a lower error rate on the query results. We will incorporate such design methodology into the extension of this work. **Lastly**, the privacy provenance table in this paper is inspired by the discretionary access control (DAC) model. It would be interesting to see in future works a more expressive model for privacy provenance, e.g., an analyst may be able to delegate his/her privacy privilege to other analysts temporarily. More research questions and works may be able to spawn by a deeper intertwinement between privacy provenance and access/leakage control [42].

Chapter 9

Conclusion

Our main research objective in this thesis is to show that a multi-analyst system interface can largely increase the utility of a differentially private SQL query system. To show this, we propose *DProvSQL*, a privacy provenance framework for differentially private SQL query engines that tracks the privacy loss to each supported data analyst. *DProvSQL* can avoid wasting privacy budgets on the same query asked by different data analysts and prevent risky queries that exceed the privilege level as per data analyst. We have designed privacy provenance table, privacy translation modules and the additive Gaussian mechanism where we incorporate them into *DProvSQL* as a middleware solution that provides privacy provenance and accuracy-aware properties to existing DP SQL query systems. The additive Gaussian mechanism uses global and local synopses to maximize query answering and utilizes the additive property of the Gaussian-related mechanisms to update synopses. We implement the prototype of *DProvSQL* and our experimental results show the efficacy and efficiency of our proposed approach.

References

[1] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. Plume: Differential privacy at scale. *arXiv preprint arXiv:2201.11603*, 2022.

[2] Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, pages 394–403. PMLR, 2018.

[3] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 87–96, 2013.

[4] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

[5] Peter Buneman and Wang-Chiew Tan. Provenance in databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1171–1173, 2007.

[6] James Cheney, Laura Chiticariu, Wang-Chiew Tan, et al. Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases*, 1(4):379–474, 2009.

[7] The Transaction Processing Performance Council. The tpc benchmark h (tpc-h)., 2008.

[8] Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Dan Kifer. Free gap information from the differentially private sparse vector and noisy max mechanisms. *Proc. VLDB Endow.*, 13(3):293–306, 2019.

[9] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2022.

[10] Wei Dong and Ke Yi. A nearly instance-optimal differentially private mechanism for conjunctive queries. *arXiv preprint arXiv:2105.05443*, 2021.

[11] Wei Dong and Ke Yi. Residual sensitivity for differentially private multi-way joins. In *Proceedings of the 2021 International Conference on Management of Data*, pages 432–444, 2021.

[12] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[13] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 486–503. Springer, 2006.

[14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[15] Cynthia Dwork and Guy N Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

[16] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 69–81, 2015.

[17] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[18] Chang Ge, Xi He, Ihab F Ilyas, and Ashwin Machanavajjhala. Apex: Accuracy-aware differentially private data exploration. In *Proceedings of the 2019 International Conference on Management of Data*, pages 177–194, 2019.

[19] Sameera Ghayyur, Dhrubajyoti Ghosh, Xi He, and Sharad Mehrotra. Mide: Accuracy aware minimally invasive data exploration for decision support. *Proceedings of the VLDB Endowment*, 15, 2022.

[20] Moritz Hardt and Guy N Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70. IEEE, 2010.

[21] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1), 2010.

[22] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia M Procopiuc, and Divesh Srivastava. Dpt: differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165, 2015.

[23] Xi He, Ashwin Machanavajjhala, and Bolin Ding. Blowfish privacy: Tuning privacy-utility trade-offs using policies. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1447–1458, 2014.

[24] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[25] Noah Johnson, Joseph P Near, Joseph M Hellerstein, and Dawn Song. Chorus: a programming framework for building scalable differential privacy mechanisms. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 535–551. IEEE, 2020.

[26] Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[27] Zach Jorgensen, Ting Yu, and Graham Cormode. Conservative or liberal? personalized differential privacy. In *2015 IEEE 31St international conference on data engineering*, pages 1023–1034. IEEE, 2015.

[28] Karl Knopf. Framework for differentially private data analysis with multiple accuracy requirements. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2890–2892, 2021.

[29] Ios Kotsogiannis, Stelios Doudalis, Sam Haney, Ashwin Machanavajjhala, and Sharad Mehrotra. One-sided differential privacy. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 493–504. IEEE, 2020.

[30] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.

[31] Fragkiskos Koufogiannis, Shuo Han, and George J Pappas. Gradual release of sensitive data under differential privacy. *arXiv preprint arXiv:1504.00429*, 2015.

[32] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. A data-and workload-aware algorithm for range queries under differential privacy. *arXiv preprint arXiv:1410.0265*, 2014.

[33] Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Steven Z Wu. Accuracy first: Selecting a differential privacy level for accuracy constrained erm. *Advances in Neural Information Processing Systems*, 30, 2017.

[34] Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi. A programming framework for differential privacy with accuracy concentration bounds. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 411–428. IEEE, 2020.

[35] Miti Mazmudar, Thomas Humphries, Matthew Rafuse, and Xi He. Cache me if you can: Accuracy-aware inference engine for differentially private data exploration. In *2020 The ACM Conference on Computer and Communications Security (CCS) Workshop on Theory and Practice of Differential Privacy (TPDP 2020)*, 2022.

[36] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.

[37] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017.

[38] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360, 2012.

[39] Arjun Narayan and Andreas Haeberlen. {DJoin}: Differentially private join queries over distributed databases. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 149–162, 2012.

[40] Joseph P Near and Xi He. Differential privacy for databases. *Foundations and Trends® in Databases*, 11(2):109–225, 2021.

[41] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, 2007.

[42] Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. Don't be a tattletale: Preventing leakages through data dependencies on access control protected data. *Proceedings of the VLDB Endowment*, 15(11), 2022.

[43] Shangfu Peng, Yin Yang, Zhenjie Zhang, Marianne Winslett, and Yong Yu. Query optimization for differentially private data management systems. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1093–1104. IEEE, 2013.

[44] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proceedings of the VLDB Endowment*, 7(8):637–648, 2014.

[45] David Pujol, Yikai Wu, Brandon Fain, and Ashwin Machanavajjhala. Budget sharing for multi-analyst differential privacy. *Proceedings of the VLDB Endowment*, 14(10):1805–1817, 2021.

[46] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment*, 6(14):1954–1965, 2013.

[47] Sofya Raskhodnikova and Adam Smith. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *arXiv preprint arXiv:1504.07912*, 2015.

[48] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

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

[50] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR, 2013.

[51] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private sql with bounded user contribution. *Proceedings on privacy enhancing technologies*, 2020(2):230–250, 2020.

[52] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering*, 23(8):1200–1214, 2010.

[53] Yingtai Xiao, Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Daniel Kifer. Optimizing fitness-for-use of differentially private linear queries. *Proceedings of the VLDB Endowment*, 14(10), 2021.

[54] Le Yu, Shufan Zhang, Lu Zhou, Yan Meng, Suguo Du, and Haojin Zhu. Thwarting longitudinal location exposure attacks in advertising ecosystem via edge computing. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022.

[55] Dan Zhang, Ryan McKenna, Ios Kotsogiannis, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. Ektelo: A framework for defining differentially-private computations. In *Proceedings of the 2018 International Conference on Management of Data*, pages 115–130, 2018.

[56] Shufan Zhang, Runchao Jiang, and Xi He. Dprovsql: Privacy provenance framework for differentially private sql engine. In *2022 The Thirty-ninth International Conference on Machine Learning Workshop on Theory and Practice of Differential Privacy (TPDP 2022)*, 2022.

[57] Wanrong Zhang, Olga Ohrimenko, and Rachel Cummings. Attribute privacy: Framework and mechanisms. *arXiv preprint arXiv:2009.04013*, 2020.