Differential Privacy for Databases

Suggested Citation: Joseph P. Near and Xi He (2020), "Differential Privacy for Databases", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Joseph P. Near
University of Vermont
jnear@uvm.edu

Xi He
University of Waterloo
xi.he@uwaterloo.ca

now
the essence of knowledge
Boston — Delft

Contents

1 Introduction **2**

2 Basics of Differential Privacy **7**
- 2.1 Definition & Properties . . . 7
- 2.2 Databases & Distance Metrics . . . 8
- 2.3 Basic Mechanisms . . . 10
- 2.4 Composition . . . 11
- 2.5 Advanced Mechanisms . . . 16

3 Problem Definition **18**
- 3.1 Queries & Query Workloads . . . 19
- 3.2 Measuring Utility . . . 20
- 3.3 Threat Model . . . 22
- 3.4 What can be Learned Accurately . . . 24
- 3.5 Additional Challenges of the Database Setting . . . 24
- 3.6 Summary of Approaches . . . 27

4 Mechanisms for Linear Queries **29**
- 4.1 MWEM . . . 30
- 4.2 Matrix Mechanism . . . 32
- 4.3 Data-Aware/Workload-Aware (DAWA) Mechanism . . . 35
- 4.4 Others . . . 37

5 Mechanisms for High-Dimensional Data **38**
5.1 DualQuery . . . 39
5.2 PrivBayes . . . 42
5.3 HDMM . . . 45
5.4 PGM . . . 47

6 Mechanisms for Highly Sensitive Queries **49**
6.1 Local Sensitivity . . . 50
6.2 Propose-Test-Release . . . 51
6.3 Smooth Sensitivity . . . 53
6.4 Sample & Aggregate . . . 55
6.5 Lipschitz Extensions . . . 57

7 Mechanisms for Multi-Relational Databases **60**
7.1 Defining Privacy for Multi-Relational Databases . . . 61
7.2 DP Systems for Multi-relational Databases . . . 65
7.3 PINQ . . . 68
7.4 Flex . . . 70
7.5 PrivateSQL . . . 73
7.6 Google DP . . . 77

8 Frameworks for Differentially Private Analysis **79**
8.1 ϵktelo . . . 81
8.2 APEx . . . 83

9 Eliminating the Trusted Data Curator **89**
9.1 The Local Model . . . 91
9.2 The Shuffle Model . . . 94
9.3 Leveraging Secure Computation . . . 95

10 Implementation Issues & Open Challenges **98**
10.1 Privacy Definitions & Algorithm Design . . . 98
10.2 System Implementation & Integration . . . 101
10.3 Social Considerations . . . 102

References **106**

Differential Privacy for Databases

Joseph P. Near[1] and Xi He[2]

[1] *University of Vermont; jnear@uvm.edu*
[2] *University of Waterloo; xi.he@uwaterloo.ca*

ABSTRACT

Differential privacy is a promising approach to *formalizing privacy*—that is, for writing down *what privacy means* as a mathematical equation. This book is provides overview of differential privacy techniques for answering database-style queries. Within this area, we describe useful algorithms and their applications, and systems and tools that implement them.

Joseph P. Near and Xi He (2020), "Differential Privacy for Databases", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

1

Introduction

Differential privacy is a promising approach to *formalizing privacy*—that is, for writing down *what privacy means* as a mathematical equation. The definition of differential privacy acts as a bridge between societal notions of privacy and the mathematical properties of privacy-preserving algorithms—we can prove that a specific algorithm satisfies differential privacy, and then argue separately that the definition is a "good" approximation of society's *informal* notions of privacy. Differential privacy has been successful because it seems to serve particularly well in this role—it is the best mathematical model of privacy that we know of.

This book is intended to serve as an overview of the state-of-the-art in techniques for differential privacy. We focus in particular on techniques for answering database-style queries, on useful algorithms and their applications, and on systems and tools that implement them. While we do describe the formal properties of the techniques we cover, our focus is not on theoretical results.

What is privacy? In this book, we use the term *privacy* to refer to situations in which an adversary is **not able to learn too much about any one individual**. When the adversary learns too much

about an individual, we say that privacy has been lost. One trivial solution for privacy is to prevent the adversary from learning *anything*—but this approach makes it pointless to collect and analyze data in the first place.

The techniques we explore in this book are ones that allow the adversary to learn *properties of the population* while hiding information specific to individuals. Such techniques allow us to learn useful information from sensitive data, while at the same time protecting the privacy of the individuals who contributed it.

What is privacy *not*? Privacy properties are often conflated with security properties. Though they are related, they are distinct in important ways. Common security properties include *confidentiality* (that an adversary learns *nothing* about the secret data) and *integrity* (that an adversary is not capable of corrupting the system's output).

Privacy-preserving algorithms do not necessarily satisfy either of these properties. Differentially private algorithms *intentionally* reveal some information to the adversary; the goal of differential privacy is to control *what can be learned* from that information.

Similarly, techniques for enforcing security properties do not necessarily ensure privacy. In particular, most techniques for security control *who* can view the data—not *what information* they can learn from it. Encrypting a dataset, for example, provides "all-or-nothing" access to its information—those without the key learn nothing, while those with the key learn everything, including information specific to individuals. Encryption, by itself, is not capable of making the distinction described above between properties of the population and properties of individuals.

However, security techniques can *complement* privacy techniques in important ways. In particular, such techniques allow us to target alternative *threat models* for differentially private algorithms. For example, many systems for differential privacy collect raw sensitive data on a central server, and assume the server will not be compromised. If the server is hacked, however, then the guarantee of differential privacy may be violated. Encrypting this data may help ensure that *only* differentially private results are ever made public—even if the server

holding the data is compromised. Complementing differential privacy thus allows us to adjust the threat model to protect against a stronger adversary than before. We discuss combining differential privacy with security techniques in Chapter 9.

Why *differential* privacy? Differential privacy is the latest in a series of approaches for building privacy-preserving algorithms. The most common technique for releasing data while preserving privacy is *de-identification* (sometimes called anonymization), which involves removing *identifying information* from the data. De-identification appeals to our intuitions about privacy, but numerous results suggest that *re-identification* attacks on de-identified data are often possible (Sweeney, 2000; Dinur and Nissim, 2003).

More rigorous techniques, like k-Anonymity (Sweeney, 2002) and ℓ-Diversity (Machanavajjhala *et al.*, 2007), were developed to address this shortcoming by quantifying the "uniqueness" of an individual within a dataset. However, even these techniques are not *compositional*—releasing a single k-Anonymized dataset might provide strong privacy protection, but releasing *two* such datasets may enable an adversary to re-identify individuals in the data.

Differential privacy is attractive because in addition to closely approximating our informal notions of privacy, it is *compositional*. Compositionality means that if two data releases *individually* provide certain levels of differential privacy, then we can bound the *cumulative* privacy loss of both releases. Differential privacy is the first rigorous approach to privacy with this important property.

What does differential privacy protect? The goal of differential privacy is to make the following promise: *if you participate in a differentially private analysis of data, you will not suffer any additional harm as a result.* Roughly speaking, the mathematical definition of differential privacy achieves this goal by requiring that the outcome of any differentially private analysis is *the same whether or not you participate* (this notion is formalized in Chapter 2).

Importantly, this guarantee does not necessarily prevent an adversary from learning details about an individual—particularly when those

details could have been learned *without the individual's participation in the analysis*. For example, if a differentially private study concludes that all people over age 50 enjoy playing tennis, then an adversary may infer that a specific 52-year-old enjoys the sport. Differential privacy does not prevent this situation, because it is possible *whether or not* the specific 52-year-old participates in the study.

What are the limits of differential privacy? A clear tension tension exists between revealing information about a dataset and protecting the privacy of its individuals—revealing too many properties of the data with too much accuracy must *necessarily* violate privacy. This idea—now often called the *database reconstruction theorem*—imposes upper bounds on what it is possible to learn before privacy is violated (Dinur and Nissim, 2003). Navigating this tension is a key part of designing differentially private algorithms, which typically have the goal of releasing the most accurate possible statistics while preserving privacy.

Why use differential privacy in database systems? Today's information systems collect and process vast amounts of data, and the majority of it flows into databases (relational or otherwise). These database systems are specifically designed to collect, store, and query data, and have been optimized for that task. If we would like to enable analysis of sensitive data with differential privacy, it is logical to develop techniques that work for database systems, because *that's where the private data is*.

However, integrating differentially private techniques with database systems presents significant challenges—many of which are discussed later in this book. In particular, a primary goal of most database systems is to abstract away execution details, so that analysts may focus on the semantics of the queries they write instead of worrying about how they will be executed. But satisfying differential privacy requires careful control over the details of how a query is executed, which sometimes breaks this abstraction.

The techniques covered in this book represent significant progress towards building differentially private database systems. They differ in

terms of their capabilities and the interfaces they present to the analyst, and none matches perfectly with the traditional abstractions used in relational databases. Indeed, significant challenges remain in achieving that goal—we discuss these in Chapter 10—and we may never get all the way there. But the approaches described in this book have already resulted in useful, deployable systems, and we hope they will pave the way towards increasing adoption of differential privacy in practice.

Summary & Additional resources. This book focuses on techniques, algorithms, and systems for answering database-style queries with differential privacy. This area is just one part of the larger field of research in differential privacy. For an introduction to the theoretical foundations of differential privacy, we refer the reader to the excellent reference by Dwork & Roth (Dwork, Roth, *et al.*, 2014). We provide additional references to more detailed descriptions of smaller sub-areas of differential privacy throughout this book.

The rest of the book is organized into three parts. The first part defines our setting and provides background: Chapter 2 describes the basics of differential privacy, and Chapter 3 describes databases and queries. Section 3.6 summarizes the specific techniques covered in the book. The second part—Chapters 4, 5, 6, and 7—describes specific techniques, categorized by application area. The third part describes progress and challenges in building differentially-private systems: Chapter 8 describes frameworks for building such systems, Chapter 9 describes the use of security techniques to support privacy, and Chapter 10 discusses implementation issues and open challenges.

2

Basics of Differential Privacy

2.1 Definition & Properties

Differential privacy (**dwork2006calibrating**) is a property of a *mechanism* $\mathcal{M}$, a randomized algorithm whose input is a *database* of records. It is written in terms of two *neighboring databases*, which are databases that differ in exactly one individual's data (more on databases and the distance between them in Section 2.2).

Definition 2.1 (Differential Privacy)**.** Let ϵ, δ be positive real number and $\mathcal{M}$ be a randomized algorithm that takes a database as input. The algorithm $\mathcal{M}$ is said to provide (ϵ, δ)-differential privacy if, for all datasets D_1 and D_2 that differ on a single element, and all $\mathcal{S} \subseteq Range(\mathcal{M})$,

$$\Pr[\mathcal{M}(D_1) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{M}(D_2) \in \mathcal{S}] + \delta, \tag{2.1}$$

where the probability is taken over the randomness used by the algorithm.

If $\delta = 0$, we say that $\mathcal{M}$ is ϵ-differentially private. The values of δ are typically set less than the inverse of any polynomial in the size of the database.

Differential privacy is immune to *p*ost-processing: A data analyst, without additional knowledge about the private database, cannot compute a function of the output of a private algorithm $\mathcal{M}$ and make it less differentially private.

Proposition 2.1. Let $\mathcal{M}$ be a randomized algorithm that satisfies (ϵ, δ)-differential privacy. Let f be an arbitrary data-independent function f which takes the output of $\mathcal{M}$ as input. Then the composed algorithm $f \circ M$ is (ϵ, δ)-differentially private.

Differential privacy can be directly extended to *group privacy* to protect the privacy of a group, for example, a family in a survey.

Theorem 2.1. Any $(\epsilon, 0)$-differentially private mechanism $\mathcal{M}$ is $(k\epsilon, 0)$-differentially private for groups of size k.

This property also holds for (ϵ, δ)-differential privacy, except the approximation term δ has a big amplification factor $ke^{(k-1)\epsilon}$ instead of k, for a group of size k.

2.2 Databases & Distance Metrics

In the differential privacy literature, a *database* $D \in \mathcal{D}$ is typically a single table of *records* (or tuples) drawn from some universe $\mathcal{X}$ (Dwork, Roth, *et al.*, 2014). Multi-relational databases are less common, though differential privacy has also been studied in this case (see Chapter 7). Databases can be represented using a **multiset of records** (or "bag of records"), but the differential privacy literature often considers other representations for convenience. Two common examples are the **histogram** representation and the **vector** representation, described below.

Differential privacy is designed around the concept of *neighboring databases*—databases which differ on the data of a single individual. The definition requires a mechanism to produce similar outcomes for neighboring database inputs, making it impossible to determine whether any particular individual participated or not.

In the most common definition of neighboring databases, we assume that each individual contributes exactly one row to the database—so

neighboring databases differ in one row. This definition has been generalized in the literature based on a *distance metric* d over the database domain (Chatzikokolakis *et al.*, 2013; Kifer and Machanavajjhala, 2014; He *et al.*, 2014a). Here we present two key differential privacy notions that are formed by different distance metrics. Two databases D_1 and D_2 are considered neighbors under a distance metric d if $d(D_1, D_2) = 1$, and the set of neighboring databases given the database domain $\mathcal{D}$ is denoted by $\mathcal{N}_d(\mathcal{D})$. Note that this set $\mathcal{N}_d(\mathcal{D})$ does not only include pairs of neighbors formed with the true instance D, but all possible databases in the domain $\mathcal{D}$.

Unbounded differential privacy. In *unbounded differential privacy*, neighboring databases are formed by **adding or removing a tuple** in the database. Under this definition, neighboring databases differ in their size. Work in this setting often models the database as a *histogram* $h(D) \in \mathbb{N}^{|\mathcal{X}|}$, in which each entry $h(D)[j]$ represents the number of elements in the database D of type $j \in \mathcal{X}$. In the histogram representation, the distance between two databases D_1 and D_2 is defined to be the l_1 distance between them, i.e., $d_{\text{unbounded}}(D_1, D_2) = \|h(D_1) - h(D_2)\|_1$.

Bounded differential privacy. In *bounded differential privacy*, neighboring databases are formed by **changing the value of exactly one tuple**. In this setting, both datasets have a fixed size n (i.e. neighboring databases are of the same size). Work in this setting often models the database as a *vector* $D \in \mathcal{X}^n$, in which $D[i]$ represents the data contributed by user i. The distance between two databases $D_1, D_2 \in \mathcal{X}^n$ is $d(D_1, D_2) = |\{i \mid D_1[i] \neq D_2[i]\}|$.

Choosing a Distance Metric. The choice of a distance metric for databases is an extremely important—and often overlooked—part of the differential privacy guarantee. Different choices for the metric can change the guarantee completely. For example, unbounded differential privacy implies bounded differential privacy, but the reverse does *not* hold. If the chosen distance metric does not accurately model the

presence or absence of an individual in the database, then even a correct differentially private mechanism may fail to protect the privacy of an individual in the real world. This issue is even more challenging in the multi-relational setting, where the presence or absence of an individual may affect many records in the database (see Chapter 7 for more).

2.3 Basic Mechanisms

2.3.1 Randomized Response

Randomized response (Warner, 1965; Dwork, Roth, *et al.*, 2014) is a simple mechanism developed in the social sciences to approximate frequency of embarrassing or illegal behaviors. This mechanism is shown differentially private. For example, given a question "do you have property P?", the respondent will respond truthfully with probability p. Hence, there is a probability $(1-p)$ to get a false response. The randomized response mechanism satisfies ϵ-differential privacy, where $\epsilon = |\ln(\frac{p}{1-p})|$. Variants of randomized response have been developed to infer answers to aggregate queries (Erlingsson *et al.*, 2014; Wang *et al.*, 2019).

2.3.2 The Laplace Mechanism

Consider numeric queries $f : \mathcal{D} \to \mathbb{R}^k$ that map databases to k real numbers. The Laplace mechanism adds noise to the query answer. An important parameter that determines the amount of noise to ensure differential privacy is the l_1-sensitivity of the query (Dwork, 2006; Dwork *et al.*, 2006b).

Definition 2.2 (l_1-sensitivity)**.** The l_1-sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^k$ is

$$\Delta_1 f = \max_{(D_1, D_2) \in \mathcal{N}_d(\mathcal{D})} \|f(D_1) - f(D_2)\|_1. \tag{2.2}$$

This parameter of query measures the largest possible change to the query answer between any pairs of neighboring databases.

Theorem 2.2 (The Laplace Mechanism)**.** Given a numeric query $f : \mathcal{D} \to \mathbb{R}^k$, the Laplace mechanism adds to the query answer $f(D)$ with

a vector $(\eta_1, \cdots, \eta_k)$, where η_i are i.i.d. random variables drawn from the Laplace distribution centred at 0 with scale $b = \Delta_1 f/\epsilon$, denoted by $Lap(b)$. The Laplace mechanism preserves $(\epsilon, 0)$-differential privacy.

2.3.3 The Gaussian Mechanism

When allowing $\delta > 0$ for differential privacy, Gaussian mechanism (Dwork *et al.*, 2006a; Dwork, Roth, *et al.*, 2014) can be applied.

Definition 2.3 (l_2-sensitivity)**.** The l_2-sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^k$ is

$$\Delta_2 f = \max_{(D_1, D_2) \in \mathcal{N}_d(\mathcal{D})} \|f(D_1) - f(D_2)\|_2. \tag{2.3}$$

Theorem 2.3 (The Gaussian Mechanism)**.** Given a numeric query $f : \mathcal{D} \to \mathbb{R}^k$, let $\epsilon \in (0, 1)$. For $c^2 > 2\ln(1.25/\delta)$, the Gaussian mechanism adds to the query answer $f(D)$ with a vector $(\eta_1, \cdots, \eta_k)$, where η_i are i.i.d. random variables drawn from $\mathcal{N}(0, \sigma^2)$, where $\sigma > c\Delta_2 f/\epsilon$. The Gaussian mechanism preserves (ϵ, δ)-differential privacy.

2.3.4 The Exponential mechanism

Given some arbitrary range $\mathcal{R}$, the exponential mechanism (McSherry and Talwar, 2007) is defined with respect to some utility function $u : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, which maps database and output pairs to utility scores. For a fixed database D, a better output from $\mathcal{R}$ should have a larger score. The sensitivity of the utility score is defined as

$$\Delta u = \max_{r \in \mathcal{R}} \max_{(D_1, D_2) \in \mathcal{N}_d(\mathcal{D})} |u(D_1, r) - u(D_2, r)|. \tag{2.4}$$

Theorem 2.4 (The Exponential Mechanism)**.** The exponential mechanism takes in the database $D \in \mathcal{D}$ and score function $u : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, and outputs an element $r \in \mathcal{R}$ with probability proportional to $\exp(\frac{\epsilon u(r, D)}{2\Delta u})$. This mechanism satisfies $(\epsilon, 0)$-differential privacy.

2.4 Composition

Differential privacy is *compositional*—that is, running a differentially private mechanism twice also satisfies differential privacy, but at increased

privacy cost. The compositionality of differential privacy separates it from a number of other privacy notions, including de-identification and k-anonymity. In both of those cases, two separate releases of data may individually satisfy the desired property, but may violate the property when taken together. Two differentially private releases of data, in contrast, may result in *increased* privacy cost, but will always satisfy differential privacy for *some* value of ϵ.

2.4.1 Sequential Composition

The *sequential composition* property of differential privacy (Dwork, 2006 says that the privacy cost of running two differentially private mechanisms on the same input data is simply the sum of their individual privacy costs.

Theorem 2.5 (Sequential Composition)**.** If there are k independent mechanisms $\mathcal{M}_1, \ldots \mathcal{M}_k$, whose privacy guarantees are $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$ differential privacy, respectively, then any function g of them: $g(\mathcal{M}_1, \cdots, \mathcal{M}_k)$ is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$-differentially private.

Sequential composition allows running multiple differentially private analyses on the same input data. For example, the U.S. Census releases numerous summaries of the data they collect, each of which can be expressed using a differentially private query on the collected data. Sequential composition allows bounding the total privacy cost for each census participant when all of these summaries are released publicly.

Sequential composition also enables the the development of more advanced privacy-preserving algorithms that use multiple differentially private mechanisms to perform their tasks. The sequential composition result holds even if the k mechanisms and their privacy budgets are adaptively chosen. For example, MWEM (discussed in Chapter 4) is an iterative algorithm that applies the Laplace mechanism many times to converge on a synthetic representation of the input data; its total privacy cost is bounded by repeated application of the sequential composition property.

Definition 2.5 provides an upper bound on the privacy cost of composing multiple *arbitrary* differentially private mechanisms, but

this bound is not necessarily tight for a *particular* combination of mechanisms. Some mechanisms (*e.g.* Report Noisy Max, discussed in Section 2.5.1) can be analyzed via sequential composition, but a more specific analysis of their behavior yields a lower privacy cost than sequential composition does.

When mechanisms are applied to disjoint subsets of data, the privacy loss is analyzed with a tighter bound (McSherry, 2009). Consider unbounded differential privacy, the total privacy loss is bounded by the *maximum* privacy budget consumed by any single one of these mechanisms. This is known as parallel composition.

2.4.2 Advanced Composition

An alternative composition property, often called *advanced composition* (Dwork, Roth, *et al.*, 2014), allows a mechanism designer to tune the tradeoff between the privacy parameters ϵ and δ. In exchange for a small increase in δ, advanced composition allows for a large decrease in ϵ. Advanced composition often yields a lower value for ϵ than sequential composition for iterative algorithms that perform a large number of differentially private operations.

Theorem 2.6 (Advanced Composition)**.** For $0 < \epsilon' < 1$ and $\delta' > 0$, the class of (ϵ, δ)-differentially private mechanisms satisfies $(\epsilon', k\delta + \delta')$-differential privacy under k-fold adaptive composition (*e.g.* a loop with k iterations) for $\epsilon' = 2\epsilon\sqrt{2k\log(1/\delta')}$.

Advanced composition requires the use of (ϵ, δ)-differential privacy (the definition requires $\delta' > 0$), so it cannot be used when ϵ-differential privacy is preferred. In addition, while advanced composition can generally provide lower values of ϵ when k is large, it is actually much worse than sequential composition (due to constant factors) when k is small (*e.g.* less than 100).

2.4.3 Variants of Differential Privacy with Improved Composition

In addition to ϵ and (ϵ, δ)-differential privacy, other variants of differential privacy with significant benefits have recently been developed. Three examples are Rényi differential privacy (RDP) (Mironov, 2017),

zero-concentrated differential privacy (zCDP) (Bun and Steinke, 2016), and truncated concentrated differential privacy (tCDP) (Bun *et al.*, 2018). Each one has different privacy parameters and a different form of sequential composition. The basic mechanism for RDP and zCDP is the Gaussian mechanism; tCDP uses the *sinh-normal* mechanism which decays more quickly in its tails. All three can be converted to (ϵ, δ)-differential privacy, allowing them to be compared and composed with each other, and are closed under post-processing. All three variants discussed in this section provide two important benefits over (ϵ, δ)-differential privacy:

- They eliminate the "catastrophic" privacy failure that is allowed with probability δ under (ϵ, δ)-differential privacy.
- They automatically provide asymptotically tight bounds on privacy cost under composition when the Gaussian mechanism is used.

Rényi Differential Privacy (RDP). Rényi differential privacy (RDP) (Mironov, 2017) is a relaxation of pure ϵ-differential privacy that works by bounding Rényi divergence. RDP implies (ϵ, δ)-differential privacy, but not the reverse; RDP admits the Gaussian mechanism, but does not allow any mechanism that results in a "catastrophic" privacy failure.

The Gaussian mechanism with $\sigma^2 = \Delta_2^2 \alpha/(2\epsilon)$ provides (α, ϵ)-RDP for a value with l_2 sensitivity Δ_2. For mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$ which provide (α, ϵ_1) and (α, ϵ_2)-RDP, respectively, their sequential composition provides $(\alpha, \epsilon_1 + \epsilon_2)$-RDP. This bound is tight for composition of the Gaussian mechanism *automatically*—no separate advanced composition theorem is needed. RDP is closed under post-processing.

Zero-Concentrated Differential Privacy (zCDP). Like RDP, zero-concentrated differential privacy (zCDP) (Bun and Steinke, 2016) leverages Rényi divergence to provide tighter bounds on composition for the Gaussian mechanism, but it considers all αs simultaneously and thus has only a single privacy parameter (called ρ). Like RDP, zCDP prevents "catastrophic" privacy failures, admits the Gaussian mechanism, and is closed under post-processing.

The Gaussian mechanism with $\sigma^2 = \Delta_2^2/(2\rho)$ provides ρ-zRDP for a value with l_2 sensitivity Δ_2. For mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$ which provide ρ_1 and ρ_2-zCDP, respectively, their sequential composition provides $(\rho_1 + \rho_2)$-zCDP. As in RDP, tight composition is automatic.

Truncated CDP (tCDP). Truncated concentrated differential privacy (tCDP) (Bun *et al.*, 2018) also bounds Rényi divergence, but relaxes the original definition of CDP to permit privacy amplification via subsampling. tCDP retains the other benefits of zCDP—tight composition, closure under post-processing, and no "catastrophic" privacy failure.

The basic mechanism of tCDP is the *sinh-normal* mechanism. For a value with l_2 sensitivity Δ_2, adding noise drawn from

$$8\Delta_2\omega \operatorname{arsinh}\Big(\frac{1}{8\Delta_2\omega}\mathcal{N}\Big(0, \frac{\Delta_2^2}{2\rho}\Big)\Big)$$

provides (ρ, ω)-tCDP. For mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$ which provide (ρ_1, ω_1) and (ρ_2, ω_2)-tCDP, respectively, their sequential composition provides $(\rho_1 + \rho_2, \min(\omega_1, \omega_2))$-tCDP. Tight composition is automatic.

Choosing a Variant. Due to their tight composition bounds, the use of these variants can reduce privacy cost significantly—especially for iterative algorithms that use the Gaussian mechanism. For example, machine learning algorithms that add Gaussian noise to calculated gradients (Abadi *et al.*, 2016) often run for thousands of iterations, and using a privacy variant based on Rényi divergence provides much tighter bounds on the privacy cost of these algorithms than either standard or advanced composition.

For algorithms that are not iterative, or that perform only a few iterations, the privacy variants discussed in this section provide little benefit over pure ϵ-differential privacy. For algorithms that use the Gaussian mechanism (and thus do not satisfy pure ϵ-differential privacy), these variants provide the benefit of eliminating the chance of a "catastrophic" privacy failure. And for iterative algorithms, these variants provide the additional benefit of significantly tighter bounds under composition.

A small handful of techniques *require* the ability to fail catastrophically (e.g. Propose-Test-Release, discussed in Chapter 6), and thus

require the use of (ϵ, δ)-differential privacy. For algorithms that do not use these techniques, the privacy variants in this section should be considered.

2.5 Advanced Mechanisms

Consider m counting queries $(f_1, \ldots, f_m)$ which has a high l_1-sensitivity equal to m at worst case. If m is large, then Laplace mechanism will add much noise to the counts. Besides applying the advanced composition techniques, there are several scenarios where we can apply specialized techniques to offer better utility.

2.5.1 Report Noisy Max

The first scenario is when the analyst is only interested in the maximum count and its index. A simple algorithm with more utility guarantees can be used. This algorithm is known as *report noisy max*. This algorithm adds independently generated Laplace noise $Lap(1/\epsilon)$ to each count and returns the index of the largest noisy count. A much smaller noise is injected into the count, but this algorithm has shown to satisfy $(\epsilon, 0)$-differential privacy.

A generalized algorithm for reporting top-k is proposed by Lee and Clifton, 2014, where Laplace noise $Lap(k/\epsilon)$ is added to each count. This algorithm also satisfies $(\epsilon, 0)$-differential privacy. Compared to the Laplace mechanism that releases all the counts of the m queries for the top-k answers, the Report Noisy Max algorithm adds much smaller noise and hence a better utility under the same privacy guarantee.

2.5.2 The Sparse Vector Technique

The second scenario is for online setting, when the analyst only care to know the identity of the counting queries that lie above a certain threshold T. A simple technique known as the *sparse vector technique* (Dwork *et al.*, 2009) supports this online scenario and its privacy loss only depends on the number of queries which actually lie above the threshold, rather than with the total number of queries tested. This

saves much privacy cost especially when the number of queries lie above the threshold is much smaller than the total number of queries.

First, we present the algorithm when only one above-threshold query is returned. This algorithm takes in a private database D, an adaptively chosen stream of counting queries $f_1, \ldots$, a threshold T. This algorithm first adds Laplace noise $Lap(2/\epsilon)$ to the threshold T to get noisy threshold $\tilde{T}$. For each query f_i, add independent Laplace noise $Lap(4/\epsilon)$ to $f_i(D)$. If the noisy answer is greater than $\tilde{T}$, then output $\top$ and halt the algorithm; otherwise, output $\perp$ and continue with the next query. This algorithm is also called *AboveThreshold* and is $(\epsilon, 0)$-differentially private.

To output more than one above-threshold query, the general Sparse Vector Technique repeats the AboveThreshold algorithm whenever an above-threshold query is reported. Consider a cutoff point k for the number of above-threshold queries. The noise added to the threshold T is $Lap(2k/\epsilon)$ and the noise added to each query answer is $Lap(4k/\epsilon)$. When k above-threshold queries are reported, the algorithm halts. This algorithm achieves $(\epsilon, 0)$-differential privacy.

3

Problem Definition

This book focuses on approaches for ensuring differential privacy in database systems. This chapter defines the common framework for these techniques, and the challenges specific to the database setting. Specifically:

- Section 3.1 defines the different kinds of *queries* supported by the techniques discussed in this book
- Section 3.2 describes how *accuracy* is measured in the setting of differential privacy for databases
- Section 3.3 describes the *threat model*—the capabilities of the adversary we are trying to protect against
- Section 3.4 discusses the kinds of queries for which *accurate answers* can typically be obtained while maintaining privacy
- Section 3.5 describes *additional challenges* specific to systems for differential privacy in the database setting
- Section 3.6 provides a roadmap for the rest of the book

3.1 Queries & Query Workloads

Generally speaking, a *query* can be any function of the database $f : \mathcal{D} \to \mathcal{A}$. Most commonly, we will be concerned with queries that output a number or vector of numbers (i.e. $f : \mathcal{D} \to \mathbb{R}^k$). In a database system, queries are typically specified using some *query language* (e.g. SQL). Languages like SQL make it easy to specify certain kinds of queries; a typical aggregation query written in a language like SQL (e.g. a `SELECT` query using the `COUNT`, `SUM`, or `AVG` aggregation functions) can be expressed as a function whose input is the database and whose output is a real number or vector of real numbers.

However, a standard SQL groupby aggregation query and hence OLAP styled queries do not directly fit this category of queries because the query answer includes not only the aggregation but also the groupby keys (or active domain values) which might be private. Database systems that support differential privacy have to distinguish the standard groupby aggregation that leaks active domain and a full domain based groupby aggregation that reports the frequency counts of the full domain values. We highlight how the database systems handle this type of aggregation in Chapter 7.

Multi-relational queries represent an important exception, since we assume that our database representation contains just a single relation. As a result, queries with joins cannot be expressed as functions of a single database. The queries involving joining face multiple challenges in defining privacy objects among the multiple relations and high global sensitivity. We discuss several approaches for extending the guarantee of differential privacy to multi-relational queries in Chapter 7.

The techniques covered here use a variety of representations for queries, from the fully general "query-as-a-function" abstraction presented above, to SQL and subsets of it, to special-purpose representations for particular classes of queries. For each technique, we therefore include a brief description of the class of queries supported and the representation used.

Linear queries. The class of *linear queries* has been the target of a considerable amount of research on differential privacy, and the techniques

discussed in Chapter 4 are specifically focused on this class. A linear query is a function of a single relation that can be expressed as a linear combination of row counts, where a "row count" refers to the number of times a particular row occurs in the relation. Many commonly-used queries are linear queries, including counting queries and summation queries.

In the "histogram" representation of a database, each element x_i of the histogram explicitly records the number of occurrences of the tuple value associated with i; a linear query is simply a linear combination of x_is. For example, to count the number of rows in the relation, we can simply assign each x_i a coefficient of 1.

Many of the techniques described in Chapter 4 use variants of the histogram representation, and represent linear queries using vectors of the coefficients for the corresponding linear combination. Thus our counting query example would be represented by a vector containing all 1s.

Query workloads. A *query workload* is simply a set of queries. In cases where the important queries are known ahead of time, it is common to record them together in a workload, and answer them all at once.

For certain classes of queries, answering the whole workload at once can yield much higher accuracy than answering each one in sequence and using sequential composition to bound the total privacy cost. This is especially true for workloads of linear queries, where the improvement in accuracy can be substantial. Most of the techniques presented in Chapter 4 follow this pattern, and are designed to answer an entire workload of linear queries at once.

3.2 Measuring Utility

The term *utility* refers to the degree to which a computed result is useful to the analyst. Utility is therefore dependent not only on the data and the function being computed, but also on *the analyst's needs*. In some cases, differentially private analysis results will be useful to the analyst even if they contain significant error; in other cases, the results need to be extremely close to the true answers in order to be useful.

Accuracy measures how close the differentially private result is to a result computed without differential privacy, and is thus independent of the analyst's needs. Most of the differential privacy literature measures accuracy as a proxy for utility, under the assumption that more accurate results will tend to be more useful for most analysts.

Accuracy bounds for differential privacy mechanisms are often given in the (α, β)-*accuracy* framework, where α represents an upper bound on absolute error and β represents the probability that this upper bound is violated.

Definition 3.1. Given a numeric query $f : \mathcal{D} \to \mathbb{R}^k$ and a differentially private mechanism $\mathcal{M}_f : \mathcal{D} \to \mathbb{R}^k$, we say that $\mathcal{M}$ is (α, β)-*accurate* if:

$$\Pr[\|f(D) - \mathcal{M}_f(D)\|_\infty \geq \alpha] \leq \beta \tag{3.1}$$

The accuracy of a differentially private mechanism is usually directly related to the privacy parameter. For example, the accuracy of the Laplace mechanism for general queries can be measured based on the fact that $\Pr[|\eta| \geq t \cdot b] = e^{-t}$, where η is drawn from $Lap(b)$ with a union bound.

Theorem 3.1. Given a numeric query $f : \mathcal{D} \to \mathbb{R}^k$, the Laplace mechanism is (α, β)-accurate for $\alpha = \ln(\frac{k}{\beta}) \cdot (\frac{\Delta_1 f}{\epsilon})$. In other words, let y be an output of Laplace mechanism that satisfies $(\epsilon, 0)$-differential privacy. Then, $\forall \beta \in (0, 1]$:

$$\Pr[\|f(D) - y\|_\infty \geq \ln(\frac{k}{\beta}) \cdot (\frac{\Delta_1 f}{\epsilon})] \leq \beta \tag{3.2}$$

The larger the privacy parameter ϵ is, the bigger distance between the true answer and the output mechanism with the same probability. This distance $\|f(D) - y\|_\infty$ also depends on the sensitivity of the function $\Delta_1 f$ and hence a highly sensitive query requires more perturbation in function output. A similar expression can be derived for the Gaussian mechanism based on the CDF of the normal distribution.

The accuracy for numeric queries is also commonly measured in terms of the expected total errors $\mathbb{E}[\|f(D) - y\|_2^2]$. Given a numeric query $f : \mathcal{D} \to \mathbb{R}^k$, the total error for the Laplace mechanism with

noise drawn from $Lap(b)^k$ is $2k \cdot b^2$ and the total error for the Gaussian mechanism with noise drawn from $\mathcal{N}(0, \sigma^2)^k$ is $k \cdot \sigma^2$.

For the exponential mechanism, the accuracy is measured to the set of outputs with the best utility score. Given the utility function $u : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ and database instance D, let $\mathrm{OPT}_u(D) = \max_{r \in \mathcal{R}} u(D, r)$.

Theorem 3.2. Given a database D, let $R_{\mathrm{OPT}_u} = \{r \in \mathcal{R} : u(D, r) = \mathrm{OPT}_u(D)\}$ denote the set of elements in $\mathcal{R}$ which attain utility score c. Let y be an output of an exponential mechanism based on u that satisfies $(\epsilon, 0)$-differential privacy, then

$$\Pr[u(D, y) \leq \mathrm{OPT}_u(D) - \frac{2\Delta u}{\epsilon}(\ln(\frac{|\mathcal{R}|}{|R_{\mathrm{OPT}_u}|}) + t)] \leq e^{-t} \tag{3.3}$$

This theorem bounds the probability that the exponential mechanism returns a "good" element of $\mathcal{R}$, where good is measured in terms of OPT_u. The result is that it will be highly unlikely that the returned element y has a utility score that is inferior to $\mathrm{OPT}_u(D)$ by more than an additive factor.

3.3 Threat Model

The definition of differential privacy is written in terms of a "database" containing many individuals' data. How this database is designed, stored, and processed in an actual system has a significant effect on the security and privacy guarantees actually achieved by the system. We introduce the *central model* of differential privacy here, since it is assumed in most of the techniques covered in this book; we discuss alternative threat models in Chapter 9.

Central model. The *central model of differential privacy* assumes the existence of a *trusted data curator*—an entity which collects sensitive data from individuals and stores all of it (without any added noise) in a database. The data curator is trusted to keep the data secure, and to execute mechanisms over it.

In the central model, the malicious party is the *analyst*—the party that receives the final output of a differentially private mechanism.

The analyst may use this output to try to infer information about the participants in the database.

The major advantage of the central model is accuracy. In the central model, data from many individuals can be aggregated *before* adding noise, which means that the scale of the noise added to the results is much smaller than the scale of the "signal" generated by aggregating the data. As we will describe in the next section, the local model is significantly worse in this regard.

The disadvantage of the central model is the requirement for a trusted data curator, which is often impossible to satisfy in practice. For example, if a company abuses access to sensitive data it collects from its customers, then we probably should not trust that company to act as a trusted data curator, even if they promise to use differential privacy. Furthermore, even trust*worthy* data curators can be subject to data breaches, which, in the central model, would result in revealing all of the sensitive data.

Other models. The most common alternative to the central model is the *local model* of differential privacy, in which each participant adds noise to their data *locally*—before it leaves their control. In the local model, the data curator collects *noisy* data from each participant; each participant's data *independently* satisfies differential privacy due to the noise added before it is collected. As a result, the differential privacy property holds in the local model even if the data curator is malicious. The downside of the local model is that it provides far less accuracy than the central model.

Other models have been developed to navigate the tradeoff between these two extremes. The *hybrid* or *shuffle* model uses an anonymous communication channel to amplify privacy, and techniques for *secure computation* apply cryptography to obtain the utility of the central model with the security of the local model. Both approaches are discussed in Chapter 9.

3.4 What can be Learned Accurately

Broadly speaking, differential privacy works best for *robust statistics*. Robust statistics are not affected much by outliers, so they are also not affected much by differential privacy's hypothetical addition or removal of a single data element. Commonly-used examples include counting, summation, and average queries. In the database setting, approaches for differential privacy are often considered in terms of which of the standard SQL aggregation functions—`COUNT`, `SUM`, `AVG`, `MIN`, and `MAX`—can be supported.

`COUNT`, `SUM`, and `AVG`. The first three aggregation functions are common targets for differentially private analysis. Counting queries are particularly well-suited: their sensitivity and accuracy properties are easy to analyze, and they often yield good accuracy. Given an upper bound on the values a data element can take (which can be enforced via top-coding), summation queries are similarly simple to handle. Average queries can be implemented as a combination of two queries: a summation and a count. All three classes are commonly implemented in systems for differential privacy (see Chapter 8 for more).

`MIN` and `MAX`. `MIN` and `MAX` queries are much more challenging under differential privacy, because the answers to these queries depend *entirely* on a single data element. As a result, producing differentially private answers to these queries while maintaining high accuracy is *impossible*, and systems for differential privacy typically do support these queries directly. Instead, some systems offer support for percentile queries or other means of estimating the scale of the data (see Chapter 8 for more).

3.5 Additional Challenges of the Database Setting

Existing relational database systems often appear to be good targets for differential privacy—at first glance, it should be possible to layer privacy protection on top of these systems without changing the interface presented to the analyst or the functionality of the system. However, this setting produces several unique challenges that have slowed the

broad adoption of differential privacy in relational databases, and deserve careful consideration in any practical deployment. We summarize the major challenges here, and provide more extensive discussion in Chapters 8 and 10.

Privacy budget. The definition of differential privacy is given in terms of a single privacy parameter ϵ, but a practical database system may answer thousands or millions of queries. The maximum cumulative ϵ allowed by the system for all of these queries is often called the *privacy budget*. The simplest approach is to add up the ϵ values for each of the queries processed by the system (by sequential composition), and stop answering queries when this sum exceeds the budget. However, this approach quickly exhausts the budget in practice. Much of the research in differential privacy is dedicated towards solving this problem—for example, the improved composition bounds mentioned in Chapter 2 improve privacy budget use directly, while the techniques for answering linear queries described in Chapter 4 aim to give better accuracy for the same value of ϵ, which improves budgeting indirectly. See Chapters 8 and 10 for more discussion.

Joins. Relational joins are extremely common in modern databases, but present a major challenge for differential privacy. In particular, producing tight bounds on the sensitivity of a query with joins is extremely challenging—the best bounds depend on the data itself, and the worst-case is *much* worse than the average-case. We discuss several state-of-the-art techniques for addressing this challenge in Chapter 7; however, the ability to answer join queries with high accuracy remains a major mismatch between the expectations of database users and the limits of our knowledge about differential privacy.

Data domains and `GROUP BY`. Techniques for ensuring differential privacy typically assume that the complete domain for each data attribute is static and known ahead of time. Database systems, on the other hand, typically do *not* enumerate the domain for each data element. For example, an analyst might write the following SQL query to construct

a histogram that counts occurrences of each element of the domain of the column `C`:

```
SELECT C, COUNT(*)
FROM T
GROUP BY C
```

The output of such a query will contain a row—a histogram "bin"—only for values of `C` that actually occur in the database. This semantics presents a challenge for differential privacy: it means that the *presence or absence of the bin itself* communicates information about the underlying data.

Systems for differential privacy typically address this issue by requiring the data curator to enumerate the domain of each data attribute, and ensure that the results of histogram queries contain bins for each value in the domain (even if the count for the value is zero). This mismatch in semantics can raise challenges in implementations and can be surprising to analysts.

Database updates. The majority of work on differential privacy assumes a static, unchanging database. The existing systems for differentially private analysis are most easily applied to static datasets; the US Census Bureau's use of differential privacy (Abowd, 2018) is a perfect example, because Census data is collected once and does not change thereafter. In practice, however, databases are often updated continuously. We discuss the handful of techniques developed for this setting in Chapter 10.

Side-channel attacks. Even when a *technique* has been proven to ensure differential privacy, its *implementation* may fail to satisfy the definition. A classic example is the floating-point vulnerability in the Laplace mechanism (Mironov, 2012): the mathematical definition of the Laplace mechanism satisfies ϵ-differential privacy, but implementations of it may not, due to imprecision in floating-point arithmetic. Timing attacks—in which a query's execution time may reveal something about the private data it analyzes—are also possible, and most systems for

Table 3.1: Approaches Covered in the Book

Name	Application	§
MWEM	Linear queries	4.1
Matrix mechanism	Linear queries	4.2
DAWA	Linear queries	4.3
DualQuery	High-dimensional data	5.1
PrivBayes	High-dimensional data	5.2
HDMM	High-dimensional data	5.3
PGM	High-dimensional data	5.4
Propose-test-release	High global sensitivity	6.2
Smooth sensitivity	High global sensitivity	6.3
Sample & aggregate	High global sensitivity	6.4
Lipschitz extensions	High global sensitivity	6.5
PINQ	Multi-relational data	7.3
Flex	Multi-relational data	7.4
PrivateSQL	Multi-relational data	7.5
ϵktelo	Framework	8.1
APEx	Framework	8.2

differential privacy do not attempt to prevent them. We discuss these issues in more detail in Chapter 8.

3.6 Summary of Approaches

The rest of this book is organized into chapters that describe techniques specific to an application area. Table 3.1 categorizes the specific techniques we cover and lists the application of each one.

- **Chapter 4** describes techniques for *workloads of linear queries* over a single table. These techniques attempt to minimize the total error over the entire workload.
- **Chapter 5** describes techniques for answering workloads of linear queries in the context of *high-dimensional data* in a single table. The best techniques in this setting differ significantly from the low-dimensional setting.
- **Chapter 6** describes techniques that work well when global

sensitivity is high (for example, queries with joins or queries over graphs).

- **Chapter 7** describes techniques specific to multi-relational data (including queries with joins). Many of these techniques adapt ideas from Chapter 6.

Finally, **Chapter 8** describes frameworks for designing differentially private systems, and **Chapter 9** discusses alternative threat models in detail. **Chapter 10** describes open challenges in the area of differential privacy.

4

Mechanisms for Linear Queries

As described earlier, a *linear query* is one that can be expressed as a linear combination of the counts of values in the database. When the database is represented by a histogram $h(D) \in \mathbb{N}^{|\mathcal{X}|}$, each entry $h(D)[j]$ is the count of tuple value $j \in \mathcal{X}$, and a linear query is one that can be expressed as a linear combination of the counts in $h(D)$.

This chapter describes differentially private algorithms for workloads of linear queries on a single database table. This setting is both relevant in practice and highly suited for differential privacy, and a number of effective algorithms exist. We describe three exemplars that characterize the space of research in this area: the Multiplicative Weights with Exponential Mechanism (MWEM) (Hardt *et al.*, 2012) algorithm, the Matrix Mechanism (MM) (Li *et al.*, 2015), and the Data-Aware/Workload-Aware (DAWA) mechanism (Li *et al.*, 2015).

These approaches work by constructing explicit histogram representations for the data, so they struggle with high-dimensional data. These algorithms are best for one- or two-dimensional data—a significant limitation in practice. Chapter 5 describes algorithms that address this limitation.

4.1 MWEM

The Multiplicative Weights with Exponential Mechanism (MWEM) algorithm (Hardt *et al.*, 2012) is a workload-aware data-dependent algorithm for constructing a differentially private synthetic approximation of a private dataset. The algorithm iteratively improves the synthetic approximation by posing differentially private queries on the private data and using the results to update the synthetic approximation with the multiplicative weights update rule.

For a workload Q of linear queries and a private database D, the algorithm initializes an approximation A randomly. Then, each iteration of the algorithm performs the following steps:

1. *Exponential mechanism:* select a query $q_i \in Q$ for which the approximation A is "worst" (i.e. $\arg\max_{q_i \in Q} |q_i(A) - q_i(D)|$) using the exponential mechanism.

2. *Laplace mechanism:* calculate a differentially private *measurement* m_i for $q_i(D)$ using the Laplace mechanism.

3. *Multiplicative weights:* update A using the measurement m_i according to the multiplicative weights update rule.

The algorithm runs for T iterations, and its output is the final value of A (or the average of its values at all iterations). The privacy of the MWEM algorithm follows from the properties of the Laplace and exponential mechanisms. If we use a privacy parameter of $\frac{\epsilon}{2T}$ for each of steps (1) and (2), then by sequential composition, the total privacy cost for the algorithm after T iterations is ϵ. The update in step (3) uses only the differentially private measurement m_i and the approximation A, so it satisfies differential privacy by post-processing.

Theorem 4.1. The MWEM algorithm satisfies $(\epsilon, 0)$-differential privacy.

Exponential and Laplace mechanisms. Step (1) is used to select a query from the workload Q for which the approximation performs badly. To do this, the algorithm uses the exponential mechanism (with privacy

parameter $\frac{\epsilon}{2T}$) with the following score function, which assigns high scores to queries for which the approximation produces high error:

$$s(D, q) = |q(A) - q(D)|$$

Step (2) simply applies the Laplace mechanism to produce a differentially private measurement of $q_i(D)$:

$$m_i = q_i(D) + \text{Lap}\Big(\frac{2T}{\epsilon}\Big)$$

Multiplicative weights update rule. Hardt *et al.* formalize the synthetic approximation A as a *distribution* over the domain $\mathcal{D}$ of records in the database. For a database D with n records, the algorithm begins by initializing A to n times the uniform distribution over $\mathcal{D}$. Then, the multiplicative weights update rule updates the weight A places on each record x according to:

$$A_{\text{new}}(x) \propto A(x) \cdot \exp(q(x) \cdot (q(D) - q(A))/2)$$

Here, $q(x)$ indicates whether the record x contributes to the answer of q ($q(x)$ is 1 if so, and 0 otherwise). The proportionality sign indicates that A should be renormalized after scaling.

Representation of A. The standard representation of A—n times a distribution over $\mathcal{D}$—is not always optimal in practice. For example, if the query workload Q contains 1-dimensional range queries, then the representation of A only needs to consider one column of the database. Hardt *et al.* give three examples of alternatives: histograms (1-dimensional), contingency tables (2-dimensional), and data cubes (n-dimensional).

In the histogram representation, the database can be represented as a function $D : \mathcal{D} \to \mathbb{R}$, and the synthetic approximation is a probability mass function $A : \mathcal{D} \to \mathbb{R}$. When $\mathcal{D}$ is discrete and finite (e.g. ages from 0 - 100), it is correspondingly easy to represent A. When $\mathcal{D}$ is continuous and infinite, a finite range can be considered and split into bins. As the dimensionality of $\mathcal{D}$ grows, however, it becomes increasingly difficult to build compact representations for A.

When to use MWEM. The MWEM algorithm is a good "default" choice for large workloads of linear queries. It is simple and easy to implement. It is efficient when the representation of A is small, and it works for arbitrary linear queries. MWEM does not require the analyst to specify queries in a particular way or require specialized transformations on queries or the database. Empirical evaluations demonstrate its effectiveness for answering query workloads with low error (Hardt *et al.*, 2012; Hay *et al.*, 2016a).

The MWEM algorithm is also flexible in its application. It can be used in an *offline* fashion, when the complete query workload Q is available ahead of time, to produce accurate answers for those queries. It can also be used in an *online* fashion, with a dynamic workload Q, to answer queries as they arrive by releasing the intermediate measurements m_i (or their approximations). In either mode, the final output of the algorithm is a synthetic approximation of the original database that can be used to answer *further* queries without additional privacy cost. The approximation will yield accurate answers for queries that are well-represented by the original workload Q (though it may not be useful for answering poorly-represented queries).

However, MWEM performs poorly in high-dimensional settings, as can A grow exponentially in the number of dimensions. In addition, MWEM often requires extensive tuning of hyperparameters (e.g. the number of iterations, and the number of queries to run per iteration) in order to achieve good performance. In general, the other techniques in this chapter tend to produce higher utility than MWEM, at the cost of greater implementation complexity.

4.2 Matrix Mechanism

The Matrix Mechanism (MM) proposed by Li *et al.*, 2015 is a workload-aware differentiallly private algorithm to answer linear counting queries. Its utility is not dependent on the input data.

Workload and data representation. The matrix mechanism begins with the observation that the answer to a query workload can be expressed as a matrix-vector product. Each linear query in the workload

can be written as a vector, and those vectors can be arranged into a matrix $\mathbf{W}$; if the database itself (in the histogram format) is written as a vector $\mathbf{x}$, then the workload's answer is simply the product $\mathbf{Wx}$.

Formally, we consider a database D as a collection of records from a universe $\mathcal{X}$ of k attributes $\{A_1, \ldots, A_k\}$ each having an ordered domain $dom(A_i)$. To express linear queries, the database instance is first transformed into a data vector of counts $\mathbf{x} \in \mathbb{N}^n$. MM mainly targets at the one- or two-dimensional case. In one dimension, let's say A_i, each entry of the data vector $\mathbf{x}[j]$ represents the number of tuples in D that take on the j^{th} value in the ordered domain of A_i. In the two-dimensional case, for attributes A_i, A_j, $\mathbf{x}$ summarizes the count for each tuple in $dom(A_i) \times dom(A_j)$. The workload of linear queries is represented as a matrix $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$ based on $\mathbf{x}$, where each linear query is a length-n row vector $\mathbf{w}$ which $\mathbf{w}[j] \in \mathbb{R}$. The answer to a linear query $\mathbf{w}$ on $\mathbf{x}$ is the dot product $\mathbf{wx} = \mathbf{w}[1]\mathbf{x}[1] + \cdots + \mathbf{w}[n]\mathbf{x}[n]$. Hence, the answer to the workload is $\mathbf{Wx}$.

The strategy matrix. It turns out that the best way answer a query workload $\mathbf{W}$ with differential privacy is often to ask a *different* set of queries, and then transform the results into a set of answers for $\mathbf{W}$. The Matrix Mechanism implements this idea using a *strategy matrix* $\mathbf{A}$ that represents a strategy for answering the queries in $\mathbf{W}$.

Given an $m \times n$ workload $\mathbf{W}$ and a query strategy matrix $\mathbf{A}$ of size $p \times n$, we say $\mathbf{A}$ supports $\mathbf{W}$ if each query in $\mathbf{W}$ can be expressed as a linear combination of queries in $\mathbf{A}$. In other words, there exists a solution matrix $\mathbf{X}$ to the linear system $\mathbf{W} = \mathbf{WA}$. This system may have multiple solutions. The Moore-Penrose pseudoinverse of matrix $\mathbf{A}$ is used to express the answer to $\mathbf{W}$, as $\mathbf{WA}^+\mathbf{y}'$, where $\mathbf{y}'$ is the noisy answer to $\mathbf{A}$. The matrix mechanism is summarized as follows.

Given an $m \times n$ workload $\mathbf{W}$, a $p \times n$ strategy matrix $\mathbf{A}$ that supports $\mathbf{W}$ and a differentially private algorithm $\mathcal{K}(\mathbf{A}, \mathbf{x})$ that answers $\mathbf{A}$ with a given database instance $\mathbf{x}$. The matrix mechanism $\mathcal{M}_{\mathcal{K},\mathbf{A}}$ outputs the following vector:

$$\mathcal{M}_{\mathcal{K},\mathbf{A}} = \mathbf{WA}^+\mathcal{K}(\mathbf{A}, \mathbf{x}) \tag{4.1}$$

Theorem 4.2. The matrix mechanism $\mathcal{M}_{\mathcal{K},\mathbf{A}}$ inherits the privacy guarantee of $\mathcal{K}$ and is unbiased if $\mathcal{K}$ is unbiased.

Examples of $\mathcal{K}$ are Laplace mechanism, Guassian mechanism, or DAWA. If $\mathcal{K}$ is a data independent algorithm, the error for the matrix mechanism has an explicit analytical bound:

$$Error_{\mathcal{K},\mathbf{A}}(\mathbf{W}) = P(\mathcal{K})\|A\|^2\|\mathbf{W}\mathbf{A}^+\|_F^2, \tag{4.2}$$

where $P(\mathcal{K})$ is a constant determined by $\mathcal{K}$. Hence, the problem is to find a query strategy $\mathbf{A}$ that supports $\mathbf{W}$ and minimizes $Error_{\mathcal{K},\mathbf{A}}(\mathbf{W})$, given a query workload $\mathbf{W}$ and a differentially private algorithm $\mathcal{K}$.

Optimizing the strategy. The utility of the Matrix mechanism depends on the strategy $\mathbf{A}$, leading to an obvious question: can we determine the *optimal* strategy for minimizing error?

Li *et al.*, 2015 shows that error-optimal strategy matrices have equal L1 column norm, which can be normalized to 1. If using a strategy with sensitivity 1, then the total error is equal to $\|\mathbf{W}\mathbf{A}^+\|_F^2$. Hence, given a workload $\mathbf{W}$, the optimization problem is summarized as below.

$$\begin{aligned} &\min_{\mathbf{A}\in\mathbb{R}^{p\times n}} \|\mathbf{W}\mathbf{A}^+\|_F^2 \\ \text{s.t.} \quad &\mathbf{W}\mathbf{A}^+\mathbf{A} = \mathbf{W}, \qquad \|\mathbf{A}\|_1 \le 1 \end{aligned} \tag{4.3}$$

However, solving this optimization problem is difficult. Li *et al.*, 2015 formulates this problem as a rank-constrained semidefinite program, which require $O(m^4(m^4+n^4))$ time to converge to the global optimum. As the dimensions of the data increases, the computation becauses infeasible in practice. The MM is extended by McKenna *et al.*, 2018 to HDMM, a differentially private mechanism for answering a workload of predicate counting queries for high-dimensional datasets. We present the details of HDMM in in Section 5.3.

When to use the Matrix Mechanism. The Matrix Mechanism is a simple and elegant framework, and can be implemented simply. The two primary challenges come in constructing the explicit representations of $\mathbf{W}$ and $\mathbf{x}$ (which can be extremely large for high-dimensional data), and in determining the best strategy $\mathbf{A}$.

For some kinds of query workloads (e.g. cumulative distribution functions), the optimal strategy $\mathbf{A}$ is well-known (Li *et al.*, 2015), and can simply be hard-coded into the implementation. In this setting, when the data is only one- or two-dimensional, the Matrix Mechanism is a good choice: it produces high utility and is simple to implement. When the query workload is more complex or less regular (e.g. when the queries are specified by an analyst), then the DAWA (§ 4.3) or HDMM (§ 5.3) approaches may work better.

4.3 Data-Aware/Workload-Aware (DAWA) Mechanism

The Data-Aware/Workload-Aware (DAWA) mechanism proposed by Li *et al.*, 2015 considers a data and workload dependent mechanism under differential privacy. Unlike the Matrix Mechanism, DAWA focuses on answering range queries. Each range query $\mathbf{w}_i$ is described by an interval $[j_1, j_2]$ on $\mathbf{x}$, for $1 \leq j_1 \leq j_2 \leq n$. The evaluation of $\mathbf{w}_i = [j_1, j_2]$ on $\mathbf{x}$ is written $\mathbf{w}_i(\mathbf{x})$ and defined as $\sum_{j=j_1}^{j_2} \mathbf{x}[j]$.

The DAWA algorithm consists of three steps: (i) private partitioning; (ii) private budget count estimation; (iii) uniform expansion. The first two steps require private interactions with the database. To ensure that the overall algorithm satisfies ϵ-differential privacy, the privacy budget is split to ϵ_1, ϵ_2 such that $\epsilon_1 + \epsilon_2 = \epsilon$ and use these two portions of the budget for the first two steps.

Stage 1. Private partitioning: This stage takes in a privacy budget ϵ_1 and the data vector $\mathbf{x}$, and outputs B, a partition of $\mathbf{x}$ into k buckets, without counts for the buckets.

Stage 2. Private bucket counting estimation: Given the partition B from the first step, this stage spends a privacy budget ϵ_2 to learn noisy estimates of the bucket counts, denoted by $\mathbf{s}$. Rather than using Laplace mechanism, a workload-aware method is used.

Stage 3. Uniform expansion: Given the noisy estimates for the k buckets, $(B, \mathbf{s})$, an estimate for the n entries of $\mathbf{x}$ is derived based on uniformity assumption: the count $\mathbf{s}[i]$ for each bucket b_i is spread uniformly amongst each position $\mathbf{x}$ that is contained in b_i. The output of this stage is the noisy data vector $\mathbf{x}'$. As this stage is a post-processing step, it still guarantees the same level of privacy. All the range queries in the workload $\mathbf{W}$ are derivable from the noisy data vector $\mathbf{x}'$.

Next, we will explain the details of the first two stages.

The first stage, private partitioning, would like to search for the optimal partition that results in the least error in the third step. It proposes a heuristic cost function for a partition: given a partition of the domain $[1, n]$ into buckets $B = \{b_1, \ldots, b_k\}$, the cost of B is defined as

$$pcost(\mathbf{x}, B) = \sum_{i=1}^{k} \sum_{j \in b_i} \left| \mathbf{x}[j] - \frac{b_i(\mathbf{x})}{|b_i|} \right| + k/\epsilon_2 \tag{4.4}$$

The first term of the cost function estimates the amount the bucket deviates from being perfectly uniform and the second term estimates the noise introduced by the second step with privacy buget ϵ_2.

Given the set of all intervals on the domain $[1, n]$, denoted by $\mathcal{B}$, DAWA first computes the cost $bcost(\mathbf{x}, b)$ for all $b \in \mathcal{B}$ and then adds noise $Z \sim Lap(2\Delta_1 bcost/\epsilon_1)$ to each cost. Last, it finds the partition with the least *noisy* cost using dynamic programming.

The cost of computing all intervals is $O(n^2 \log n)$ and can be reduced to $O(n \log^2 n)$ by restricting to intervals whose length is a power of two. The searching of the optimal partition requires time linear in n and the number of buckets.

As this stage keeps the noisy counts secret and only publishes the partition with the least (noisy) cost, a small amount of noise is sufficient to ensure privacy, similar as the *report noisy max* algorithm described in Section 2.5.1. The noise added to the cost per bucket is proportional to the sensitivity of the bucket cost. By setting $\Delta_1 bcost \leq 2$, the private partition step can be shown ϵ_1-differentially private.

The second stage, private bucket count estimation, takes in the partition $B = \{b_1, \ldots, b_k\}$ from the first stage of DAWA, and privately estimate the bucket counts using the remaining privacy budget ϵ_2. Instead of adding Laplace noise directly to the true bucket counts $\mathbf{s}$, DAWA uses a workload-aware approach. First, DAWA transforms the given workload $\mathbf{W}$ that is based on the original domain $[1, n]$ to $\hat{\mathbf{W}}$ such that the answer to the workload $\mathbf{Wx}$ can be estimated by the product of the transformed workload $\hat{\mathbf{W}}$ and the bucket counts. Then it applies the Matrix Mechanism described in Section 4.2 to answer the transformed workload $\hat{\mathbf{W}}$. As DAWA focuses on range queries, it

only searches among hierarchical strategies — a tree of queries over buckets B to get the strategy that minimizes error for answering $\hat{\mathbf{W}}$. After obtaining the optimal strategy $\mathbf{A}$, DAWA answers this strategy matrix with Laplace mechanism using privacy budget ϵ_2 and then use the answer to the strategy matrix to reconstruct the bucket counts .

This stage takes $O(mk \log k + k^2)$ time, where m is the number of queries in $\mathbf{W}$ and k is the number of buckets in B. The privacy guarantee of this step is the same as the Matrix Mechanism, which guarantees ϵ_2-differential privacy.

By sequential composition, DAWA satisfies differential privacy with total privacy cost $(\epsilon_1 + \epsilon_2)$.

Theorem 4.3. The DAWA mechanism satisfies $(\epsilon_1 + \epsilon_2, 0)$-differential privacy.

When to use DAWA. The decision about when to use DAWA is easier to make than in other cases: the algorithm is specifically designed for range queries on one- or two-dimensional data, and often offers better utility than other approaches in this specific setting. For other kinds of queries, or for higher-dimensional data, DAWA is not a suitable choice.

4.4 Others

Many additional algorithms have been proposed for answering workloads of linear queries with differential privacy. The DPComp study (Hay *et al.*, 2016b) provides a comprehensive list and comparison of these approaches.

5

Mechanisms for High-Dimensional Data

The approaches described in Chapter 4 work well for low-dimensional data, but have scalability or accuracy challenges when used for high-dimensional data. In this setting, algorithms must carefully address both scalability and accuracy. This chapter discusses several state-of-the-art approaches for answering workloads of linear queries in the high-dimensional setting.

Conceptually, these approaches work by building differentially private estimates of *marginal distributions* over the private data. A *marginal distribution* is a probability distribution over a *subset* of a collection of random variables (in our setting, this collection is the set of attributes in the database). The marginal distribution of a subset of k attributes is often called a k-way marginal.

We begin with a simple approach for estimating marginal distributions with differential privacy, listed in Algorithm 1. This algorithm estimates the marginal distribution of a subset of a database's attributes by counting the number of occurrences of each tuple in the domain of the subset, adding noise to each count, and then normalizing the counts to sum to 1. Algorithm 1 satisfies ϵ-differential privacy by the Laplace mechanism; the l_1 sensitivity of building "count" structure is 1.

Algorithm 1: Simple Algorithm for Private Marginals

Input : Dataset $D \in \mathcal{D} = \mathcal{P}(\mathcal{X})$ where $\mathcal{X} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, target set of attributes $A \subseteq \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$.

Output : A *private marginal*: a probability distribution $\mathsf{Prob}[t]$ over tuples $t \in \mathcal{X}_A$ ($\mathcal{X}$ restricted to attributes in A).

for $t \in \mathcal{X}_A$ **do**
  $\mathsf{count}(t) \leftarrow |\{t' \in D \mid t \text{ and } t' \text{ agree on the attributes in } A\}|$
$\mathsf{noisyCount}(t) \leftarrow \max(0, \mathsf{count}(t) + \mathsf{Lap}(\frac{1}{\epsilon}))$
$\mathsf{total} \leftarrow \sum_{t \in \mathcal{X}_A} \mathsf{noisyCount}(t)$
for $t \in \mathcal{X}_A$ **do**
  $\mathsf{Prob}[t] \leftarrow \frac{\mathsf{noisyCount}(t)}{\mathsf{total}}$
return Prob

Algorithm 1 produces accurate results when the set of target attributes A is small. As A grows in size, the size of $\mathcal{X}_A$ also grows, and the number of tuples in each "bin" of "count" shrinks. When these counts are small enough, the noise added for differential privacy overwhelms the count.

This tradeoff is challenging for high-dimensional data. When the set of attributes is large, we cannot simply set A to be the complete set—the resulting marginal distribution would be extremely noisy. For a dataset with n attributes, we could use Algorithm 1 to compute n accurate 1-way marginals for a total privacy cost of $n \cdot \epsilon$, but this approach would not capture correlations between attributes.

One of the major challenges in the high-dimensional setting is thus to determine correlations between attributes and determine how to estimate a probability distribution over the entire dataset using lower-dimensional marginals. Each of the approaches in this chapter takes a different approach to solving this problem.

5.1 DualQuery

The DualQuery algorithm (Gaboardi *et al.*, 2014) has the same goal as MWEM—it is an iterative, workload-aware, data-dependent algorithm for constructing a synthetic approximation of a private dataset. The

Inputs: query workload Q, private data D, number of iterations T, privacy parameter ϵ
Output: synthetic database of T records

1 :	Sample s queries $q_1, \ldots, q_s$ from Q	*Pick s queries randomly*
2 :	$x_1 \leftarrow \arg\max_x \frac{1}{s} \sum_{j=1}^{s} q_j(x)$	*Generate an initial record*
3 :	**for** $i \leftarrow 2, \ldots, T$ **do** :	*Run T iterations...*
4 :	$u_i(D, q) = \sum_{j=1}^{i-1} q(D) - q(x_j)$	*Define the score function*
5 :	**for** $j \leftarrow 1, \ldots, s$ **do** :	*Sample s queries...*
6 :	$q_j \leftarrow \text{ExpMech}(D, u_i, Q, \epsilon)$	*Select a "bad" query*
7 :	$x_i \leftarrow \arg\max_x \frac{1}{s} \sum_{j=1}^{s} q_j(x)$	*Generate a new record*
8 :	Return $\bigcup_{i=1}^{T} x_i$	*Release the synthetic records*

Figure 5.1: The DualQuery Algorithm (Simplified)

algorithm's name comes from the fact that the algorithm is conceptually the *dual* of MWEM. Instead of maintaining and improving a distribution over data elements, DualQuery maintains a distribution over *queries* (which can be represented compactly) and uses an external solver to generate a single concrete data record at each iteration.

Since the universe of data records can be extremely large, especially for high-dimensional data, maintaining a distribution over data records is costly; this is the limiting factor for scalability in algorithms like MWEM. Since DualQuery does not maintain such a representation, it is capable of scaling to high-dimensional data. In DualQuery, the computationally challenging step—generating a new data record—can be done without access to the private data, and can therefore be outsourced to a standard constraint solver (e.g. CPLEX).

A simplified version of the DualQuery algorithm appears in Figure 5.1. Our simplifications are designed to expose the important concepts behind the algorithm; see Gaboardi *et al.*, 2014 for the full version. The private data is provided to the algorithm as a database $D \in \mathcal{D}$. In our simplified version, each query q in the query workload Q returns a single number (i.e. $q : \mathcal{D} \to \mathbb{R}$). The algorithm returns a set of T *synthetic records* $(x_1, \ldots, x_T) \in \mathcal{D}$.

The algorithm builds a list of synthetic data records $x_1, \ldots, x_T$ representing the synthetic database. It begins with an initialization step: the algorithm randomly picks s queries from the workload Q (line 1), and generates (using CPLEX) the first record x_1 by maximizing the value of the selected queries on x_1 (line 2). At each iteration, the algorithm uses the exponential mechanism to sample s queries from the workload Q (lines 5-6). The score function used in the exponential mechanism (line 4) assigns a high score for queries whose *outputs are too low* on the synthetic records $x_1, ..., x_{i-1}$ (compared with the private data D); the goal is then to generate new synthetic records to *increase those outputs*. To accomplish this, the algorithm generates (using CPLEX) a new data record x_i that maximizes the value of the selected queries. After all T iterations have finished, the algorithm outputs the union of all T generated records. Note that the algorithm has no method for *decreasing* the output value of any query—so our simplified variant will only work well if we start from an empty set of synthetic records and use queries with non-negative outputs on the true private data.

The full version of the algorithm (see Gaboardi *et al.*, 2014) includes an update rate η and target accuracy metrics α and β. The optimization steps in lines 2 and 7 are relaxed to allow the solver to return "good enough" data records for achieving the desired accuracy, which improves solving time. The full algorithm also replaces the explicit use of the exponential mechanism with a sampling step from an equivalent distribution over queries.

Privacy. The only use of the private data D is via the exponential mechanism (line 6); the computationally challenging steps (generating new data records in lines 2 and 7) do not involve the private data, so they can be easily outsourced to an external solver. The algorithm involves $s(T-1)$ uses of the exponential mechanism, for a total privacy cost (by sequential composition) of $s\epsilon(T-1)$. The sensitivity of the score function u_i is complicated slightly by the fact that it performs a summation; for iteration i, if each query $q \in Q$ is 1-sensitive, the score function has sensitivity $\Delta u_i = i - 1$. Note that the sensitivity of the score function increases with the iteration number.

Theorem 5.1. The DualQuery algorithm satisfies $(\epsilon, 0)$-differential privacy.

Properties. As mentioned earlier, the algorithm does not maintain a distribution over the universe of data records, so its scalability is limited only by the solver's ability to generate new data records. In an empirical evaluation, Gaboardi et al. demonstrate the use of DualQuery to generate synthetic approximations for datasets of up to 17,770 dimensions.

Unlike MWEM, DualQuery is not useful in an online setting, because it does not compute accurate measurements for queries of interest during its execution. Accurate answers for queries in the workload can only be obtained after all iterations have been completed.

Finally, DualQuery tends to produce less accurate answers than MWEM, when MWEM is applicable (see Gaboardi *et al.*, 2014). The strength of DualQuery is its ability to handle higher-dimensional data, rather than the accuracy of its results.

5.2 PrivBayes

PrivBayes proposed by Zhang *et al.*, 2014 is a differentially private algorithm that publishes high-dimensional data. This algorithm aims to approximate the set of high-dimensional distribution from a set of carefully chosen set of data-dependent low-dimensional marginals. To achieve this goal, PrivBayes considers the following structure for a database with attribute set $\mathcal{A} = \{A_1, \ldots, A_d\}$.

A *Bayesian network* $\mathcal{N}$ on $\mathcal{A}$ is a directed acyclic graph (DAG) over these attributes and models conditional independence among attributes using directed edges. This network $\mathcal{N}$ is defined as a set of d attribute-parent pairs, $(X_1, \Pi_1), \ldots, (X_d, \Pi_d)$, such that (i) each X_i is a unique attribute in $\mathcal{A}$; (ii) the parent set of X_i in $\mathcal{N}$, Π_i, is a subset of the attributes in $\mathcal{A} \backslash \{X_i\}$; (iii) for any $1 \leq i \leq j \leq d$, we have $X_j \notin \Pi_i$, i.e., there is no edge from X_j to X_i in $\mathcal{N}$ — this ensures that the network is acyclic. The *degree* of $\mathcal{N}$ is defined as the maximum size of any parent set Π_i in $\mathcal{N}$.

Let $\Pr[\mathcal{A}]$ denote the full distribution of tuples in database D. This joint probability can be approximated with d conditional distributions based on the attribute-parent pairs in $\mathcal{N}$, i.e.,

$$Pr[\mathcal{A}] = Pr[X_1, \ldots, X_k] = \prod_{i=1}^{d} \Pr[X_i|\Pi_i] \tag{5.1}$$

A good approximation of $\Pr[\mathcal{A}]$ requires that (i) $\mathcal{N}$ accurately captures the conditional independence among the attributes in $\mathcal{A}$; (ii) the d conditional distributions are accurately learned from the data. If $\mathcal{N}$ have a small degree, then the learning of the conditional distributions is simply learning of d low-dimensional joint distributions $(\Pr[X_1, \Pi_1], \ldots, \Pr[X_d, \Pi_d])$.

Based on this idea, PrivBayes consists of the following three steps.

Step 1. Network learning: This step takes in a privacy budget $\epsilon/2$ and the private data to construct a k-degree Bayesian network $\mathcal{N}$ over attributes in D (k is a small value that can be chosen automatically by PrivBayes). PrivBayes takes a greedy approach to add attribute-parent pairs into the network. First, it randomly picks an attribute X_1 from $\mathcal{A}$ and adds $(X_1, \emptyset)$ to an empty network $\mathcal{N}$. Next, among all possible attribute-parent sets (X, Π) where X is from the remaining attributes and Π is a subset of the attributes that have been added to the current $\mathcal{N}$, PrivBayes applies exponential mechanism to pick the "best" pair and add it to $\mathcal{N}$. This process repeats till all attributes in $\mathcal{A}$ have an attribute-parent pair in $\mathcal{N}$.

The first-cut exponential mechanism for picking the best attribute-parent pair considers a popular score function in a non-private setting, the *mutual information* between attributes $I(X, \Pi)$, but this score function is highly sensitive and allows the exponential mechanism to output a poor choice of attribute-parent pair. Hence, PrivBayes define a new notion *maximum joint distribution* for an attribute-parent pair (X, Π) as *the joint distribution that maximizes the mutual information between* X *and* Π, denoted by $\Pr^{\diamond}[X, \Pi]$. This new function has a small sensitivity $\frac{1}{n}$ and it is also a good replacement of $I(X, \Pi)$ — if $\Pr^{\diamond}[X, \Pi]$ is large, then $I(X, \Pi)$ tends to be large. The computation time for this new function is $O(n2^k)$, but in practice, PrivBayes only consider low-degree Bayesian network, and hence, the algorithm is still practical.

Step 2. Distribution learning: The remaining privacy budget $\epsilon/2$ is spent to compute the necessary differentially private distributions of the data in the subspaces of the Bayesian network. Let the maximum degree of the network $\mathcal{N}$ be k. This step first materializes the joint distribution $\Pr[X_i, \Pi_i]$ for $i \in [k{+}1, d]$ and then perturb them by Laplace noise with scale $4(d-k)/(n\epsilon)$. Based on the noisy joint distribution $\Pr^*[X_i|\Pi_i]$, we can obtain all the conditional distributions $Pr[X_i|\Pi_i]$ for all $i = 1, \ldots, d$. This step satisfies $(\epsilon/2, 0)$-differential privacy.

Step 3. Data synthesis: We can approximate the distribution of the tuples in D using the noisy distributions, i.e., $\Pr^*[\mathcal{A}] = \prod_i^k \Pr^*[X_i|\Pi_i]$ and then sample tuples from $\Pr^*[\mathcal{A}]$. However, this is very costly. Instead, PrivBayes perform sampling without materializing $\Pr^*[\mathcal{A}]$, but sample the attributes of each tuple in an increasing order or i, such that we will be able to sample X_j from $\Pr^*[X_j|\Pi_j]$ given the previous sampled attributes. In this way, a synthetic database D^* with the same size as D is generated.

Privacy. Step 1 and step 2 spent half of the privacy budget each and by sequential composition, they achieve $(\epsilon, 0)$-differential privacy. Step 3 is simply a post-processing step that uses the private network structure and the joint distribution to sample records.

Theorem 5.2. The PrivBayes algorithm satisfies $(\epsilon, 0)$-differential privacy.

Properties. The paper shows a utility guarantee for the noisy distributions learned in step 1 – the ratio of average scale of information to average scale of noise is no less than $\frac{n\cdot\epsilon}{(d-k)\cdot 2^{k+3}}$. In addition, the paper evaluated the PrivBayes algoirthm on two types of tasks: (i) build all α-way marginals of a dataset; and (ii) simultaneously train multiple SVM classifiers on a dataset, where each classifier predicts one attribute in the data based on all other attributes.

5.3 HDMM

The high-dimensional matrix mechanism (HDMM) proposed by McKenna *et al.*, 2018 extends the basic Matrix Mechanism by Li *et al.*, 2015 (Section 4.2) for answering a workload of predicate counting queries for high-dimensional datasets. This approach exploits a compact representation to efficiently search (a subset of) the space of differentially private algorithms for one that answers the input query workload with high accuracy. HDMM first represents the query workloads $\mathbf{W}$ using an implicit matrix representation $\mathbb{W}$. Then it select the optimized strategy $\mathbb{A}$ using this implicit representation $\mathbb{W}$. Next, it applies Laplace mechanism to get noisy answer $\mathbf{y}$ for this strategy $\mathbb{A}$ and infers answers to the implicitly vectorized data $\mathbf{x}'$ from $(\mathbb{A}, \mathbf{y})$. Last, return answer $\mathbb{W}\mathbf{x}'$. This algorithm satisfies the same privacy guarantee as the Laplace mechanism. Readers can refer to Section 4.2 for the details of the Matrix Mechanism. In this section, we will focus on (i) the compact representation of the workload and the data, and (ii) the accuracy optimization based on the compact representation.

Implicit Representations. In the Matrix Mechanism, a data vector $\mathbf{x}$ represents the count for each tuple in the full domain $dom(A_1) \times \cdots dom(A_k)$. A predicate counting query ϕ on $\mathbf{x}$ is represented by a length-n vector $\mathbf{w}$, where entry equal to $\phi(t) \in \{0, 1\}$ for each tuple $t \in dom(A_1) \times \cdots \times dom(A_k)$. Given a predicate $[\phi]_{A_1}$ defined on a single attribute A_1, where $\|dom(A_1)| = n_1$, we can vectorize this predicator with respect to just the domain of A_1 instead of the full domain of all attributes. For a predicate that is formed from the conjunction of such single attribute predicates, its vectorized form has a concise implicit representation in terms of the *kronecker product*, denoted by $\otimes$, between vectors. A length n vector is treated as an $1 \times n$ matrix.

Definition 5.1 (Kronecker Product)**.** For two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{m' \times n'}$, their Kronecker product $A \otimes B \in \mathbb{R}^{m \cdot m' \times n \cdot n'}$ is:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} \cdots a_{1n}\mathbf{B} \\ \vdots \quad \ddots \quad \vdots \\ a_{m1}\mathbf{B} \cdots a_{mn}\mathbf{B} \end{bmatrix}$$

Theorem 5.3 (Implicit Vectorization). Let $\phi = [\phi_1]_{A_1} \wedge [\phi_2]_{A_2}$ be a predicate defined by the condjuction of ϕ_1 and ϕ_2 on attributes A_1 and A_2. Then, $vec(\phi) = vec(\phi_1) \otimes vec(\phi_2)$.

The explicit representation of $vec(\phi)$ has size $n_1 \cdot n_2$, but the implicit representation, $vec(\phi_1) \otimes vec(\phi_2)$ requires storing only $vec(\phi_1)$ and $vec(\phi_2)$ of size $n_1 + n_2$. This representation can be generalized to a set of predicates.

Theorem 5.4. Given predicate sets $\Phi = [\phi_1, \ldots, \phi_p]_{A_1}$ and $\Psi = [\psi_1, \ldots, \psi_r]_{A_2}$, on attributes A_1 and A_2, the vectorized product is defined in terms of matrices $vec(\Phi)$ and $vec(\Psi)$: $vec(\Phi \times \Psi) = vec(\Phi) \otimes vec(\Psi)$.

Given a set of predicate counting queries on the full domain $\mathbf{W} = \{q_1, \ldots, q_k\}$ and weights $w_1, \ldots, w_k$, HDMM proposes a way to convert them into an implicit representation workload matrix. For each $q_i \in \mathbf{W}$, where $q_i = [\Phi_{i1}]_{A_1} \times \cdots \times [\Phi_{id}]_{A_d}$, its implict representation is

$$\mathbb{W}_i = vec(\Phi_{i1}) \otimes \cdots \otimes vec(\Phi_{id}).$$

Hence, the implicit workload is represented as

$$\mathbb{W}_{[k]} = \begin{bmatrix} w_1\mathbb{W}_1 \\ \vdots \\ w_k\mathbb{W}_k \end{bmatrix} = \begin{bmatrix} w_1(vec(\Phi_{11}) \otimes \cdots \otimes vec(\Phi_{1d})) \\ \vdots \qquad\qquad \ddots \qquad\qquad \vdots \\ w_k(vec(\Phi_{k1}) \otimes \cdots \otimes vec(\Phi_{kd})) \end{bmatrix} \tag{5.2}$$

Besides saving the storage cost, this implicit representation of the workload accelerates the key steps of the Matrix Mechanism. First, the matrix product can be computed using the implicit representation. Given a matrix with its implicit representation $\mathbb{W} = \mathbf{W}_1 \otimes \cdots \mathbf{W}_d$, we have $\mathbb{W}^T\mathbb{W} = \mathbf{W}_1^T\mathbf{W} \otimes \cdots \mathbf{W}_d^T\mathbf{W}$. Second, a critical step is to compute the pseudoinverse of the strategy matrix $\mathbf{A}$ on the full domain. Given the implicit representation of $\mathbf{A}$ as $\mathbb{A} = \mathbf{A_1} \otimes \cdots \otimes \mathbf{A_d}$, its pseudoinverse can be implicitly represented as $\mathbb{A}^+ = \mathbf{A}_1^+ \otimes \cdots \otimes \mathbf{A}_d^+$. Lastly, the sensitivity of the strategy matrix can be calculated as $\|\mathbb{A}\|_1 = \prod_{i=1}^d \|\mathbf{A}_i\|_1$.

Optimization. Recall that the goal of the Matrix Mechanism in Section 4.2 is to find a strategy $\mathbf{A}$ that (a) supports the input workload $\mathbf{W}$ while (b) offering minimum expected total squared error (Equation 4.3).

HDMM limits the search space of the strategy matrix to a small class of strategies, *p-identity matrix*:

$$\mathbf{A}(\Theta) = \begin{bmatrix} \mathbf{I} \\ \mathbf{\Theta} \end{bmatrix} \mathbf{D} \tag{5.3}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{D} = diag(\mathbf{1}_N + \mathbf{1}_p\mathbf{\Theta})^{-1}$. $\mathbf{D}$ is a diagnoal matrix that scales the columns of $\mathbf{A}(\Theta)$ so that $\|\mathbf{A}\| = 1$. The $p \times n$ values in $\mathbf{\Theta}$ determines the weights of the p queries as well as the weights on the identity queries (where a lower weight query will be answered with greater noise). The optimization problem becomes parameterized in terms of $\mathbf{\Theta}$:

$$\min_{\mathbf{A}\in\mathbf{A}(\mathbf{\Theta})|\mathbf{\Theta}\in\mathbb{R}_+^{p\times n}} \|\mathbf{W}\mathbf{A}^+\|_F^2. \tag{5.4}$$

This optimization problem does not guarantee the optimal strategy matrix, but it reduces the number of variables and allow more effective gradient-based optimization. Given strategy $\mathbf{A}$, the objective function, denoted by $C(\mathbf{A})$, and the gradient function, denoted by $\frac{\partial C}{\partial \mathbf{A}}$, are defined as

$$C(\mathbf{A}) = \|\mathbf{W}\mathbf{A}^+\|_F^2 = tr[(\mathbf{A}^+\mathbf{A})(\mathbf{W}^+\mathbf{W})] \tag{5.5}$$

$$\frac{\partial C}{\partial \mathbf{A}} = -2\mathbf{A}(\mathbf{A}^T\mathbf{A})^+(\mathbf{W}^T\mathbf{W})\mathbf{A}^T\mathbf{A}^+ \tag{5.6}$$

Given any p-Identity strategy $\mathbf{A}(\mathbf{\Theta})$, both the obejctive function and the gradient can be evaluated in $O(pn^2)$ time. Despite the performance improvement over MM, this parametized optimization is only feasible for special workloads and low-dimensional data.

To address high-dimensional data, a series of optimization is used in HDMM based on the implicit representation of the workloads. The key idea is to decompose a strategy optimization problem on a multi-dimensional workload into a sequence of optimization problems on individual attributes.

5.4 PGM

Probabilistic graphical-model (PGM) based estimation and inference is applied by McKenna *et al.*, 2019 to improve the accuracy and scalability

of many state-of-the-art mechanisms. Differing from all the differential private mechanisms introduced earlier, PGM is a post-processing step, and hence costs no additional privacy budget. It can be applied as a pluggable subroutine that could be used within the three prior techniques (DualQuery, PrivBayes, and HDMM).

Let $\mathbf{y}$ be a set of answers for workload queries $\mathbf{W}$ outputted by an ϵ-differentially private mechanism (e.g. DualQuery, PrivBayes, HDMM). The goal of PGM is to derive answers to a possibly different workload queries $\mathbf{W}'$, given $\mathbf{y}$. Past works first estimate a full contingency table $\hat{\mathbf{p}}$ from the given measurements $\mathbf{y}$ that minimizes a loss function $L(\mathbf{p}) = \|\mathbf{W}\hat{\mathbf{p}} - \mathbf{y}\|$ and then infer the new query answers $W'\hat{\mathbf{p}}$. However, this approach does not scale to high dimensions since the size of the full contingency table is exponential in the dimensions while the first workload queries $\mathbf{W}$ are often over low-dimensional queries.

Instead of estimating the full contingency table, PGM considers a graphical model $\mathbf{p}_\theta(\mathbf{x}) = \frac{1}{z}\exp(\sum_{C\in\mathcal{C}}\theta_C(\mathbf{x}_C))$ over marginals $\mathbf{x}_C$ to model the data distribution of the input data $\mathbf{x}$. Given the set of marginals associated to $\mathbf{y}$, PGM searches the best parameter vector θ such that the graphical model $\mathbf{p}_\theta(\mathbf{x})$ has maximum entropy among all $\hat{\mathbf{p}}$ with these marginals. In this way, the structure of the graphical model is determined by $\mathbf{y}$, such that no information is lost compared to a full contingency table representation. Moreover, its representation is much more compact when expressing the measurements in $\mathbf{y}$ over a low-dimensional marginal. After $\hat{\mathbf{p}}_\theta$ has been estimated, PGM uses it to infer answers to new queries without materializing the full contingency table representation.

Readers can refer to the paper for the details. It is shown that incorporating PGM estimation significantly reduces the error. For example, for PrivBayes, worklaod errors can be improved up by 7x and for DualQuery, an error reductions up to 4.4x. Moreover, incorporating PGM estimation into MWEM and HDMM enables these algorithms feasible on the larger datasets and workloads.

6

Mechanisms for Highly Sensitive Queries

The techniques described so far have all leveraged *global sensitivity* (defined in Definitions 6.1 and 2.3), which considers all possible pairs of neighboring databases. For some queries, global sensitivity is unbounded, because the maximum difference in the query's output on two neighboring databases *depends on the data*.

One common example is counting triangles in a graph. An example of such a graph is shown in Figure 6.1; it contains two triangles (ABC and BCD). We might describe this graph in a database using one row per edge; in this setting, neighboring databases represent graphs differing in exactly one edge.

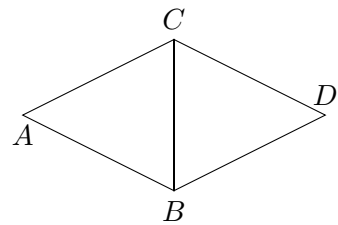

Figure 6.1: A graph that induces unbounded global sensitivity for triangle-counting.

Consider what happens if we remove the edge BC—the resulting graph has *no* triangles. This is a neighboring database to the one in Figure 6.1—we have removed a single edge from the graph—but the query's output changed by 2. If three triangles contained the edge BC, then the output would

change by 3.

It turns out that for any bound we might posit on the global sensitivity of the triangle-counting query, it is possible to construct a graph that violates the bound. The global sensitivity of the triangle-counting query is *unbounded*.

A triangle-counting query can be considered as a self-join query on a table of edges. In general, queries involving joint operators encounter unbounded global sensitivity. Even simple aggregates like median of an attribute can have a high sensitivity. Adding or removing a record can change the median from the smallest possible value in the domain to the largest possible value. Hence, it is important to design algorithms for highly sensitive queries.

There are two common approaches for handling queries like these under differential privacy: *local sensitivity*, described next and used in three of the four techniques described in this chapter, and *truncation* or *Lipschitz extension*, described in Section 6.5.

6.1 Local Sensitivity

Local sensitivity (Nissim *et al.*, 2007) is similar to global sensitivity, except that it considers only neighbors of the *true database* being considered (instead of considering *all pairs* of neighbors).

Definition 6.1 (l_1 local sensitivity)**.** The l_1 local sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^k$ on database instance x is

$$LS_1(f, x) = \max_{(x, D_2) \in \mathcal{N}_d(\mathcal{D})} \|f(x) - f(D_2)\|_1.$$

Note that x is the actual database being used to run the query f, and the max quantifies only over D_2. The l_2 local sensitivity is obtained by replacing the l_1-norm in the definition by the l_2-norm.

The local sensitivity of the triangle-counting query mentioned earlier is bounded, because it considers only neighbors of the *actual graph* we are querying. For the example graph in Figure 6.1, the local sensitivity of the triangle-counting query is 2—by adding or removing an edge in *that graph*, we can change the query's output by at most 2 triangles.

Unlike global sensitivity, local sensitivity depends on the database (in addition to the query). As a result, local sensitivity *is itself sensitive*—it might reveal information about the data. Nissim et al. provide a compelling example using a median query, in which local sensitivity can reveal the median itself. Because of this challenge, approaches have been developed that allow the use of local sensitivity without leaking information about the data; we discuss three of them in the next sections.

6.2 Propose-Test-Release

The Propose-Test-Release (PTR) framework (Dwork and Lei, 2009) allows releasing results with Laplace noise calibrated to local sensitivity with (ϵ, δ)-differential privacy. The framework has three steps:

1. The analyst *proposes* a sensitivity b.
2. The framework performs a differentially private *test* to check that the true database is "far" from one with local sensitivity higher than b.
3. If the test passes, the results of the query are *released* with Laplace noise scaled to $\frac{b}{\epsilon}$.

Clearly, b does not reveal anything about the data, because the analyst has proposed it. The check that the true database is far from one with local sensitivity higher than b is *itself* differentially private, so the information leakage from the test can be bounded. The PTR approach thus sidesteps the problem of local sensitivity leaking information by *applying differential privacy.*

PTR satisfies (ϵ, δ)-differential privacy (instead of pure ϵ-differential privacy) because there is a chance the test in step (2) passes "by accident." The test is itself randomized (to satisfy differential privacy), and the noise in the test can cause it to pass even though b is not a "good" proposal. In this case, the noise added in step (3) is not actually sufficient to satisfy ϵ-differential privacy. The δ parameter for PTR quantifies the chances that the test passes "by accident."

To define PTR formally, we need a way to measure how "far" a database is from one with high local sensitivity. We adopt the following definition (Nissim *et al.*, 2007) of the *local sensitivity at distance* k, which measures the following: imagine you add or remove k records in the database x, in a *worst-case* manner that *maximizes* the local sensitivity of the new database you construct (we say that the new database is "at distance k from x"). Then measure the local sensitivity of that new database.

Definition 6.2 (Local Sensitivity at Distance)**.** (Nissim *et al.*, 2007, Definition 3.1). The local sensitivity of f at distance k from database x is:

$$A_f^{(k)}(x) = \max_{d(x,y)\leq k} LS_1(f, y)$$

Theorem 6.1 (Propose-test-release)**.** (adapted from Dwork and Lei, 2009). The following mechanism satisfies (ϵ, δ)-differential privacy:

1. **Propose:** Propose a target bound b on local sensitivity.
2. **Test:** Let $d = \arg\min_k(A_f^{(k)}(x) > b)$.
   If $d + \mathsf{Lap}(\frac{1}{\epsilon}) \leq \frac{\log(2/\delta)}{\epsilon}$, return $\perp$.
3. **Release:** Return $f(x) + \mathsf{Lap}(\frac{b}{\epsilon})$.

The key to the privacy of PTR is that the distance d has a *global* sensitivity of 1, so adding Laplace noise to d with the scale $\frac{1}{\epsilon}$ satisfies ϵ-differential privacy. The right-hand side of the test is defined so that the test succeeds by accident with probability at most δ (by the fact that if $Y \sim \mathsf{Lap}(b)$, then $\Pr[|Y| \geq t \cdot b] = \exp(-t)$).

When to use Propose-Test-Release. PTR is relatively easy to apply, and conceptually easy to understand. The major challenge is typically calculating $A_f^{(k)}$—doing so directly would seem to require calculating local sensitivity for *all possible databases* at distance k. For this reason, most applications of PTR use an efficiently-computed upper bound on $A_f^{(k)}$ that is domain-specific (i.e. works only for a specific class of queries f). We will see some examples in Chapter 7.

6.3 Smooth Sensitivity

An alternative approach for safely leveraging local sensitivity is *smooth sensitivity* (Nissim *et al.*, 2007). Smooth sensitivity works by ensuring that the sensitivity is "smooth enough" that the noise used to achieve privacy also prevents information leakage from the sensitivity itself. To accomplish this, Nissim et al. use the concept of β-*smoothness* to quantify how smooth the sensitivity is.

Definition 6.3 (β-smoothness (Nissim *et al.*, 2007))**.** A function f is β-smooth if for all neighboring databases x and y:

$$f(x) \leq e^{\beta} f(y)$$

To apply smooth sensitivity, we will need to show that the measure of sensitivity we want to use is β-smooth, and that it is an upper bound on local sensitivity.

Definition 6.4 (Smooth bound on local sensitivity (Nissim *et al.*, 2007, Definition 2.1))**.** For $\beta > 0$, a function $S : \mathcal{X}^n \to \mathbb{R}$ is a β-smooth upper bound on the local sensitivity of f if:

- $S(x)$ is an upper bound on $LS_1(f, x)$: $\forall x \in \mathcal{X}^n.\ S(x) \geq LS_1(f, x)$
- $S(x)$ is β-smooth: $\forall x, y \in \mathcal{X}^n, d(x, y) = 1.\ S(x) \leq e^{\beta} \cdot S(y)$

One way to achieve this definition is to *force* the function to be β-smooth, by explicitly examining the local sensitivity of nearby databases. If we find one with very high local sensitivity, then our smooth upper bound should use that local sensitivity instead, "discounted" according to the distance between the true database and the one with high local sensitivity.

Definition 6.5 (Smooth sensitivity (Nissim *et al.*, 2007, Definition 2.2))**.** For $\beta > 0$, the β-smooth local sensitivity of f is:

$$S^*_{f,\beta}(x) = \max_{y \in \mathcal{X}^n} \left(LS_1(f, y) \cdot e^{-\beta d(x,y)} \right)$$

In practice, the value $S^*_{f,\beta}(x)$ is virtually impossible to calculate directly, because it requires considering the local sensitivity of *all* databases (no matter how far they are from the true database). Instead, practical applications of smooth sensitivity typically use a *function-specific* β-smooth upper bound on local sensitivity that can be efficiently calculated. Nissim et al. give examples for the median, minimum, cost of a minimum spanning tree, and triangle-counting functions.

Calibrating noise to smooth sensitivity. Achieving differential privacy using a β-smooth upper bound on local sensitivity requires scaling noise in a slightly different way than we saw earlier for global sensitivity. Nissim et al. present a framework of (α, β)-*admissible* noise distributions to specify these requirements. Here, we describe several useful examples—for noise drawn from the Cauchy, Laplace, and Gaussian distributions.

Lemma 6.2 (Cauchy distribution (Nissim *et al.*, 2007, Lemma 2.7))**.** For $\gamma > 0$, if $S(x)$ is a β-smooth upper bound on the local sensitivity of f for $\beta = \frac{\epsilon}{2d(\gamma+1)}$ and Z is sampled from the generalized (d-dimensional) Cauchy distribution with density $h(z) \propto \frac{1}{1+|z|^\gamma}$, then the following mechanism satisfies $(\epsilon, 0)$-differential privacy:

$$\mathcal{M}(x) = f(x) + \frac{2(\gamma+1) \cdot S(x)}{\epsilon} \cdot Z$$

Lemma 6.3 (Laplace distribution (Nissim *et al.*, 2007, Lemma 2.9))**.** If $S(x)$ is a β-smooth upper bound on the local sensitivity of f for $\beta = \frac{\epsilon}{4(d+\ln(2/\delta))}$ and Z is sampled from the d-dimensional Laplace distribution, then the following mechanism satisfies (ϵ, δ)-differential privacy:

$$\mathcal{M}(x) = f(x) + \frac{2S(x)}{\epsilon} \cdot Z$$

Lemma 6.4 (Gaussian distribution (Nissim *et al.*, 2007, Lemma 2.10))**.** If $S(x)$ is a β-smooth upper bound on the l_2 local sensitivity of f for $\beta = \frac{\epsilon}{4(d+\ln(2/\delta))}$ and Z is sampled from the d-dimensional Gaussian distribution, then the following mechanism satisfies (ϵ, δ)-differential privacy:

$$\mathcal{M}(x) = f(x) + \frac{5\sqrt{2\ln(2/\delta)} \cdot S(x)}{\epsilon} \cdot Z$$

When to use Smooth Sensitivity. Because it is challenging to apply, smooth sensitivity is not as commonly used as global sensitivity-based approaches. As mentioned earlier, efficiently bounding smooth sensitivity can be extremely challenging. In addition, smooth sensitivity requires extra noise compared to global sensitivity—the definitions above all have a constant factor of at least 2—so using smooth sensitivity usually does not make sense when it is possible to bound global sensitivity.

In addition, the definition of smooth sensitivity is tied to a specific definition of neighboring databases—the one associated with *bounded differential privacy* (see Section 2). Smooth sensitivity, as originally defined, cannot be used to achieve unbounded differential privacy.

However, smooth sensitivity can be extremely useful when there is no way to bound global sensitivity directly. In these cases, *all* approaches for achieving differential privacy require making tradeoffs, and the requirements of smooth sensitivity may be more reasonable than the alternatives. We will see an example of its use in practice in Chapter 7.

6.4 Sample & Aggregate

The *Sample & Aggregate* framework (Nissim *et al.*, 2007) is another way to take advantage of local sensitivity, and does not require computing local sensitivity directly. For a query function $f : \mathcal{D} \to A$ and a differentially private aggregation function $\mathcal{M} : \mathcal{P}(A) \to \mathbb{R}^d$, the Sample & Aggregate framework takes the following steps:

1. **Sample:** Split the database D into k disjoint chunks $D_1, \ldots, D_k$.
2. **Run f:** Calculate the query's answer on each chunk: $a_i = f(D_i)$ for $i \in 1, \ldots, k$.
3. **Aggregate:** Release the result of running the aggregation mechanism $\mathcal{M}$ on the answers: $\mathcal{M}(a_1, \ldots, a_k)$.

Note that since the chunks $D_1, \ldots, D_k$ constructed in step (1) are disjoint, each record in D will appear in *exactly one chunk*. This means that each record influences exactly one answer a_i—so adding or removing a record in the input database will cause at most one of the a_is to change.

Instantiating $\mathcal{M}$. One common way to instantiate the aggregation function $\mathcal{M}$ is with a differentially private mean. If the expected range of outputs of f is known ahead of time, we can use the a *clipped mean* mechanism, which satisfies ϵ-differential privacy:

1. **Clip answers:** Let $a'_i = \max(lowerBound, \min(upperBound, a_i))$.
2. **Noisy sum:** Let $s = \left(\sum_{i=1}^{k} a'_i\right) + \mathsf{Lap}\left(\frac{upperBound - lowerBound}{\epsilon}\right)$.
3. **Noisy mean:** Release $\frac{s}{k}$.

In many cases, it is difficult to bound the output range of f tightly. In these cases, an adaptive algorithm can be used to find the mean (e.g. Smith, 2011). However, such algorithms are often slower and less accurate than the simple clipped mean.

When to use Sample & Aggregate. The primary benefit of Sample & Aggregate is that no bound on the sensitivity of the query f is required—it works for *any* f, no matter how sensitive. In particular, Sample & Aggregate can work well when f's sensitivity depends on the data—especially when f's worst-case sensitivity is high, but its average-case sensitivity is low (for example, when calculating the median). Sample & Aggregate can take advantage of low average-case sensitivity in these cases.

The challenge in applying Sample & Aggregate lies in dealing with the potentially uncertain *output range* of f. In effect, the framework shifts the burden from proving a bound on the sensitivity of f, to estimating the range of f's output.

If f's output definitely lies in a certain range (e.g. a machine learning classifier with a finite set of output classes), then the clipped mean can be easily applied. However, many important queries do not have this property, and instead have data-dependent output ranges (e.g. the median). In these cases, Sample & Aggregate can be effectively applied if the analyst already knows something about the range of the data (e.g. upper and lower bounds), but it can be very challenging to apply otherwise.

The privacy of Sample & Aggregate is easy to see: we argue above that exactly one of the a_i's differs between two neighboring databases, regardless of f's sensitivity, so if $\mathcal{M}$ satisfies ϵ-differential privacy, then the whole framework does too. It is not so easy to prove accuracy bounds for the framework, since accuracy depends heavily on the data.

6.5 Lipschitz Extensions

When the global sensitivity of a query is unbounded, it is sometimes possible to modify the query to *force* it to have bounded global sensitivity. One way of formalizing these transformations is via *Lipschitz extensions* (Kasiviswanathan *et al.*, 2013b; Blocki *et al.*, 2013; Raskhodnikova and Smith, 2015).

Definition 6.6 (Lipschitz constant (Raskhodnikova and Smith, 2015))**.** Let $f : X \to Y$ be a function from a domain X to a range Y with associated distance measures d_X and d_Y. Function f has Lipschitz constant c (equivalently, is c-Lipschitz) if $d_Y(f(x), f(x')) \leq c \cdot d_X(x, x')$ for all $x, x' \in X$.

The notion of a Lipschitz constant can be seen as a generalization of global sensitivity to arbitrary distances. If a function f is c-Lipschitz, then its global sensitivity is bounded by c.

Definition 6.7 (Lipschitz extension (Raskhodnikova and Smith, 2015))**.** Consider a domain X and a range Y with associated distance measures d_X and d_Y, and let $X' \subset X$. Fix constants $c > 0$ and $s \geq 1$. Given a c-Lipschitz function $f' : X' \to Y$, a function $f : X \to Y$ is a Lipschitz extension of f' from X' to X with stretch s if

- f is an extension of f', that is, $f(x) = f'(x)$ on all $x \in X'$ and
- f is $s \cdot c$-Lipschitz.

If $s = 1$, then we call f a Lipschitz extension of f' from X' to X (omitting the stretch).

In the context of differential privacy, the idea of a Lipschitz extension is to find a subset of all databases (X') such that the desired query

(f') has low global sensitivity on the "good" subset, then construct a transformed query (f) that gives the same output on the "good" databases while maintaining low global sensitivity over *all* databases.

Applying the Lipschitz extension. The function f in Definition 6.7 has bounded global sensitivity, due to the definition of c-Lipschitz. Hence, if we know a way to transform a function f' into a c-Lipschitz extension f, then we can achieve differential privacy for f' by running f instead and adding noise scaled to its global sensitivity.

The challenge of applying the Lipschitz extension idea comes in transforming f' into f. A common strategy is *clipping*—we construct f from f' as follows:

$$f(x) = f'(clip(x))$$

Here, the *clip* function transforms a value $x \in X$ to a value $clip(x) \in X'$ (i.e. *clip* has the type $X \to X'$). The key observation is that many functions f' are *already* c-Lipschitz, if the right clipping function is chosen. For example, if f' is the triangle-counting query described at the beginning of this Chapter, then we can use a *clip* function which ensures the degree of each node in the graph is bounded by a constant c (e.g. by throwing out edges beyond this bound). With this definition of *clip*, the construction above is indeed a Lipschitz extension of the triangle counting query; it has global sensitivity bounded by c, and we can achieve differential privacy for f in the usual way (e.g. by adding Laplace noise scaled to $\frac{c}{\epsilon}$).

When to use Lipschitz extensions. The idea of Lipschitz extensions (and associated ideas of clipping) are used in many other mechanism definitions, including some that we will discuss in Chapter 7. Lipschitz extensions are typically used in this way: as a building block in constructing larger mechanisms that serve a specific purpose. In particular, the construction of the Lipschitz extension depends heavily on the application, and no single approach for this construction works well in all cases.

The primary advantage of this approach is the ability to leverage global sensitivity rather than local sensitivity. The use of global sensitiv-

ity simplifies the privacy analysis of the larger mechanism and usually results in less total noise.

The main challenge in constructing a good Lipschitz extension comes in minimizing the amount of information thrown away by the extension. Our example construction above actually discards information in the underlying data—in that case, by throwing out edges beyond a bound. In our example, setting c small may reduce the function's sensitivity, but it also results in potentially more information thrown away. Achieving good accuracy using this approach therefore requires a careful balancing of the scale of the noise against the quantity of information retained by the extension. This balancing act is especially tricky because the optimal setting often *depends on the data*. If this tradeoff can be successfully navigated, then approaches based on Lipschitz extensions can often add less noise than the other techniques described in this chapter and thus produce more accurate results.

7

Mechanisms for Multi-Relational Databases

The mechanisms we have seen so far are limited to databases of a single table. This chapter discusses approaches for answering queries over *multi-relational* databases—databases that include multiple tables. This setting is particularly challenging for two main reasons.

First, defining neighboring databases and hence privacy for multiple tables depends on the constraints between the tables. A naive approach is to join the multiple tables into a single table and apply differentially private mechanisms for a single table. This approach does not only increase the storage cost and the query processing time in many cases, but also fails to provide sufficient privacy protection for some private tables. For example, consider the database of Figure 7.1 with schema Person(pid, age, hid) and Household(hid, st, type). Person.pid is a foreign key to Household. The single table representation of this database only supports neighboring databases that differ in a row of a person, not a row of a household. Hence, a differentially private mechanism for the Person-Household table does not provide bounded privacy guarantee for the household.

Second, multi-relational queries containing relational joins have *unbounded* global sensitivity. Consider the same example in Figure 7.1.

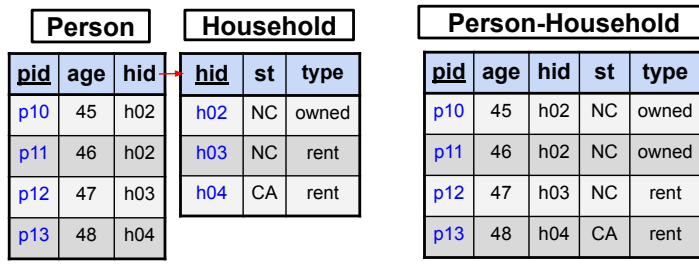

Figure 7.1: A multi-relational database under foreign key constraints (left); a single table database (right)

If a row of the Household table is removed, the foreign key relationship between Household and Person suggests that all the persons associated with this household should also be removed. If there is no bound on the maximum household size, then there is no bound on the number of persons removed in such an operation, and the sensitivity of a query on the Person table will be unbounded.

This chapter first introduces differential privacy definitions for multi-relational databases, and then describes three different methods for handling highly sensitive queries that involve joins.

7.1 Defining Privacy for Multi-Relational Databases

To design differential privacy mechanisms for multi-relational databases, we first need to define the meaning of differential privacy for them, and also specify the set of admissible queries over multi-relational databases. We will formalize these notions using the notation of Kotsogiannis *et al.*, 2019.

A multi-relational database consists of *relations* $\mathbb{S} = (R_1, \ldots, R_k)$, and each relation has a set of attributes $attr(R_i)$. We will denote the domain of an attribute $A \in attr(R)$ by $dom(A)$, and the domain of a set of attributes $\mathbf{A} \subseteq attr(R)$ as $dom(\mathbf{A}) = \Pi_{A \in \mathbf{A}} dom(A)$. The domain of a relation R is $dom(attrs(R))$.

Neighboring databases. To define differential privacy for a multi-relational database, we must first define the notion of neighboring

databases. The decision about how to do this has important consequences for the privacy property that results, and several different definitions exist. We begin with the simplest (and most limited) definition of neighboring databases, used by PINQ (§ 7.3) and Flex (§ 7.4): neighboring databases have the same set of relations and attributes, and differ in exactly *one row* of *one relation*. We can define this notion for both *unbounded* and *bounded* differential privacy (introduced in Chapter 2).

Definition 7.1 (One-row neighbors for multi-relational databases—unbounded differential privacy)**.** Two multi-relational databases D_1 and D_2 with the same set of relations (and attributes) $(R_1, \ldots, R_k)$ are *one-row neighbors* under **unbounded differential privacy** if:

- D_1 contains exactly one more record than D_2
- There exists exactly one relation $R_i \in \{R_1, \ldots, R_k\}$, and exactly one record $r \in R_i$, such that $r \in D_1$ but $r \notin D_2$
- All other records in D_1 and D_2 are identical

The PINQ system (§ 7.3) uses this definition for neighboring databases, and targets unbounded differential privacy.

Definition 7.2 (One-row neighbors for multi-relational databases—bounded differential privacy)**.** Two multi-relational databases D_1 and D_2 with the same set of relations (and attributes) $(R_1, \ldots, R_k)$ are *one-row neighbors* under **bounded differential privacy** if:

- D_1 and D_2 have the same number of records
- There exists exactly one relation $R_i \in \{R_1, \ldots, R_k\}$ such that D_1 and D_2 differ in exactly one record in R_i (i.e. the record that differs does not "switch relations")
- All other records in D_1 and D_2 are identical

The Flex system (§ 7.4) uses this definition, and targets bounded differential privacy. Neither of these definitions is capable of capturing

the Person-Household foreign key constraint mentioned at the beginning of this chapter; instead, under these definitions, we may construct a neighboring database by removing a Household without removing the corresponding persons from the Person table.

Beyond one-row neighbors: multi-relations with constraints. The PrivateSQL system (§ 7.5) uses a richer notion of neighboring databases based on database-specific privacy policies to specify privacy notions at multiple resolutions. The neighboring definition considers key constraints within databases, denoted by $\mathcal{C}$, in particular primary and foreign key constraints. When one row of a relation is removed, multiple rows from another table need to be removed due to the presence of a foreign key constraint.

A *key* is an attribute A or a set of attributes $\mathbf{A}$ that act as the *primary key* for a relation to uniquely identify its rows. We denote the set of keys in a relation R by $\mathsf{keys}(R)$. A foreign key is a key used to link two relations.

Definition 7.3. Given relations R, S and primary key A_{pk} in R, a foreign key can be defined as:

$$S.A_{fk} \rightarrow R.A_{pk} \equiv S \; {}_{A_{fk}}\ltimes_{A_{pk}} R = S$$

where the semijoin is the multiset $\{s \mid s \in S, \exists r, s[A] = r[B]\}$. That is, for every row in $s \in S$ there is exactly one row $r \in R$ such that $s[A_{fk}] = r[A_{pk}]$. We say that row $s \in S$ *refers* to row $r \in R$ ($s \rightarrow r$), and that relation S refers to relation R ($S \rightarrow R$). The attribute (or set of attributes) A_{fk} is called the foreign key.

A set of k tables $\mathbf{D} = (D_1, \ldots, D_k)$ is called a valid database instance of $(R_1, \ldots, R_k)$ under the schema $\mathbb{S}$ and constraints $\mathcal{C}$ if $\mathbf{D}$ satisfies all the constraints in $\mathcal{C}$. All valid database instances under $(\mathbb{S}, \mathcal{C})$ are denoted by $dom(\mathbb{S}, \mathcal{C})$.

Given a database relational schema $\mathbb{S}$, PrivateSQL defines a *privacy policy* as a pair $P = (\mathtt{R}, \epsilon)$, where $\mathtt{R}$ is a relation of $\mathbb{S}$ and ϵ is the privacy loss associated with the entity in $\mathtt{R}$. The relation $\mathtt{R}$ is denoted as the *primary private relation*. The output of a mechanism enforcing $P = (\mathtt{R}, \epsilon)$ does not significantly change with the addition/removal of rows in $\mathtt{R}$.

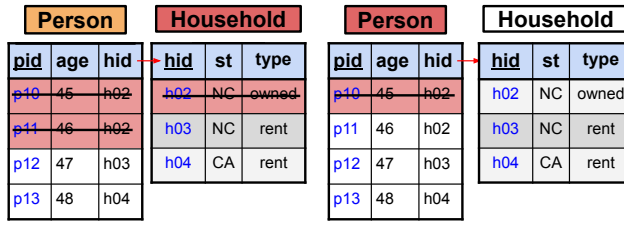

Figure 7.2: Neighboring database instances under Household Policy (left); Neighboring database instances under Person Policy (right).

To capture privacy policies and key constraints, PrivateSQL defines a neighbors inspired by Blowfish privacy (He *et al.*, 2014b). For two database instances $\mathbf{D}$ and $\mathbf{D}'$, we say that $\mathbf{D}$ is a *strict superset* of $\mathbf{D}'$ (denoted by $\mathbf{D} \sqsupset \mathbf{D}'$) if (a) $\forall i, D_i \supseteq D'_i$ and (b) $\exists i, D_i \supset D'_i$. That is, all records that appear in $\mathbf{D}'$ also appear in $\mathbf{D}$ and there is at least one row in a relation of $\mathbf{D}$ that does not appear in $\mathbf{D}'$.

Definition 7.4 (Neighboring Databases)**.** Given a schema $\mathbb{S}$ with a set of foreign key constraints $\mathcal{C}$, and a privacy policy $P = (R_i, \epsilon)$, for a valid database instance $\mathbf{D} = (D_1, \ldots, D_k) \in dom(\mathbb{S}, \mathcal{C})$, we denote by $\ominus_{\mathcal{C}}(\mathbf{D}, R_i)$ a set of databases such that $\forall \mathbf{D}' \in \ominus_{\mathcal{C}}(\mathbf{D}, R_i)$:

- $\exists r \in D_i$, but $r \notin D'_i$, and
- $\mathbf{D}'$ satisfies $\mathcal{C}$, and
- $\nexists \mathbf{D}''$ that satisfies $\mathcal{C}$ and $\mathbf{D} \sqsupset \mathbf{D}'' \sqsupset \mathbf{D}'$.

That is, $\mathbf{D}'$ is a valid database instance that results from deleting a minimal set of records from $\mathbf{D}$, including r. We call database instances $\mathbf{D}, \mathbf{D}'$ neighboring databases w.r.t. relation R_i if $\mathbf{D}' \in \ominus_{\mathcal{C}}(\mathbf{D}, R_i)$.

For example, consider the database instances from Figure 7.1 that has a foreign key constrain between Person table and Household table. We show two types of privacy policy in Figure 7.2. The pair of neighboring tables are under Person privacy policy $P = (Person, \epsilon)$. When person p10 is removed, the Household table is unchanged. However, under the Household privacy policy $P = (Household, \epsilon)$, removing h02 from the Household table results in deleting two rows in the Person table.

For this case, neighboring databases differ in both the primary private relation Household as well as a secondary private relation Person.

Definition 7.5 (DP for Multiple Relations with Constraints)**.** Given a schema $\mathbb{S}$ with foreign key constraints $\mathcal{C}$ and privacy policy $P = (\mathtt{R}, \epsilon)$ be a policy. A mechanism $\mathcal{M} : dom(\mathbb{S}, \mathcal{C}) \rightarrow \Omega$ is P-differentially private if for every set of outputs $O \subseteq \Omega$, $\forall \mathbf{D} \in dom(\mathbb{S}, \mathcal{C})$, and $\forall \mathbf{D'} \in \ominus_{\mathcal{C}}(\mathbf{D}, \mathtt{R})$:

$$\left|\ln\left(Pr[\mathcal{M}(\mathbf{D}) \in O] / \Pr[\mathcal{M}(\mathbf{D'}) \in O]\right)\right| \leq \epsilon$$

Most variants of differential privacy that apply to relational data can be captured using a single private relation and foreign key constraints on an acyclic schema (Arapinis *et al.*, 2016; Chen and Zhou, 2013; Karwa *et al.*, 2011; Kasiviswanathan *et al.*, 2013a; Dwork *et al.*, 2010; Lu *et al.*, 2014). For instance, a graph $G = (V, E)$ can be represented as a schema with relations Node(id) and Edge(src_id, dest_id) with foreign key references from Edge to Node (src_id $\rightarrow$ id and dest_id $\rightarrow$ id). Edge-DP (Karwa *et al.*, 2011) is captured by P-DP by setting Edge as the primary private relation R, Node-DP (Kasiviswanathan *et al.*, 2013a) is captured if we set Node as R. Under the latter policy, neighboring databases differ in one row from Node and all rows in Edge that refer to the deleted Node rows. Similarly, user-level- and event-level-DP are also captured using a database schema User(id, ...), Event(eid, uid, ...) with events referring to users via a foreign key (uid $\rightarrow$ id). By setting the Event (User) as the primary private relation, we get Event-DP (User-DP, resp.) (Dwork *et al.*, 2010).

7.2 DP Systems for Multi-relational Databases

Existing DP systems for multi-relational databases vary in their privacy guarantees, supported queries, and utility optimization techniques. We summarize four representative systems from the literature in Table 7.1. First, for the privacy guarantees, PINQ system (McSherry, 2009) and Flex (Johnson *et al.*, 2018) consider one-row neighbor for multi-relational databases. Flex currently develops algorithms that supports bounded DP. GoogleDP (Wilson *et al.*, 2020) considers key constraints from a user table to other tables and hence offers only one

Table 7.1: DP Systems for Multi-relational Databases

System	Neighbors	Class of queries	Bounding sens.
PINQ	One-row	Grouping, equijoins with group keys	Grouping first before join
GoogleDP	Constraints (single policy)	Grouping, semijoins on foreign key, no correlated subqueries	Manually specified bound + sampling
Flex	One-row	Grouping, equijoins including self-joins, semijoins, some correlated subqueries	Smooth sensitivity
PrivateSQL	Constraints (multiple policies)	Grouping, equijoins including self-joins, semijoins, correlated subqueries	Truncation + automatically learned threshold

type of privacy policy while PrivateSQL (Kotsogiannis *et al.*, 2019) considers multiple possible privacy policies over neighboring databases with foreign key constrains (e.g. person-policy or housedhold policy in Figure 7.2).

PINQ and GoogleDP support a very limited class of queries. Flex supports a larger class of queries summarized in Figure 7.3. Only counting queries with grouping (with user-defined group keys) are supported; the language does not include subqueries or other aggregation functions. Equijoins are supported, via queries of the form:

```
SELECT COUNT(*) FROM T1, T2 WHERE a = b
```

In addition, PrivateSQL can support equijoins and subqueries in the where clause, which can be correlated to attributes in the outer query. For example,

```
SELECT COUNT(*) FROM T1 WHERE
  (SELECT COUNT(*) from T2 WHERE a=b)=2
```

The complete query grammar can be found in the full paper of PrivateSQL (Kotsogiannis *et al.*, 2019).

Note that as the standard SQL GROUPBY operator leaks the active domain. Hence, all the database systems that support DP have to handle

$$
\begin{array}{rcl}
A & \in & \text{Attributes} \\
V & \in & \text{Constants} \\
T & \in & \text{Tables} \\
\\
Q & ::= & \textsf{SELECT } A^*, \textsf{COUNT(*) FROM } T^+ \textsf{ WHERE } Exp \textsf{ GROUPBY } A^* \\
Exp & ::= & A \; Op \; V \mid A \; Op \; A \mid Exp \textsf{ AND } Exp \mid Exp \textsf{ OR } Exp \\
Op & ::= & = \mid > \mid <
\end{array}
$$

Figure 7.3: Simple Queries for Multi-Relational Databases. GROUPBY operators require user-defined keys.

this operator carefully. In particular, when grouping is at the end of a query, then PINQ, Flex, and PrivateSQL require analyst-specified group keys. Google DP uses a slightly different approach, described in Section 7.6.

The major challenge of supporting queries over multi-relational databases is that global sensitivity is *unbounded* for counting queries that involve joins (much like the triangle-counting query in Chapter 6). The join of two tables T_1 and T_2 finds records in the two tables with matching values for the join key; multiple rows in T_2 might match a single row in T_1, and each match results in a row in the output of the join. Adding a new row to T_1 might therefore result in *many* new rows in the output of the join—the number of new rows *depends on the values in* T_2.

PINQ (Section 7.3) supports only a restricted form of joins, that allows linking unique records. GoogleDP samples a manually specified number of samples to limit the sensitivity of a query. Flex (Section 7.4) uses smooth sensitivity (Nissim *et al.*, 2007) to bound the sensitivity of join queries via local sensitivity. PrivateSQL (Section 7.5) truncates database tables before joining, to enforce an upper bound on the global sensitivity of joins.

As PINQ, Flex, and PrivateSQL share similar rules for their sensitivity analysis, we summarize their rules in Table 7.2. Each one provides a bound on *sensitivity* (for queries themselves) and on *stability*, which bounds the difference in the outputs of relational transformations (e.g. a join) run on two neighboring multi-relational databases (a formal definition is given in the next section). The stability of a sequence of

transformations is used to bound the sensitivity of a counting query over the transformations' results. The first key difference is that FLEX tracks a data-dependent sensitivity/stability in the table, and hence we see x appears in the function; while PINQ and PRIVATESQL tracks the global sensitivity/stability bound. Another difference is how FLEX and PRIVATESQL handle join operators. In particular, PRIVATESQL keeps track of the keys and hence obtains tighter sensitivity/stability bounds. The specifics of each approach are discussed in the following sections.

7.3 PINQ

Privacy Integrated Queries (PINQ) proposed by McSherry, 2009 is a platform that answers SQL-like queries on databases with differential privacy guarantee. This platform is built on top of LINQ declarative query language. The techniques are generalizable to any SQL-like queries. For each query received, the platform automatically analyzes the query and then perturbs the query answer with the right amount of noise.

Stability. Each SQL-like query is treated as a sequence of database transformation operators, like SELECTION, GROUPBY, and JOIN. Each transformation is associated with a *stability* property that controls the privacy loss over the input databases.

Definition 7.6 (Transformation stability)**.** A transformation T is said to be c-stable if for any two input data sets A and B,

$$|T(A) \oplus T(B)| \leq c \times |A \oplus B|, \tag{7.1}$$

where $\oplus$ represents the symmetrical difference between two multi-sets.

For instance, SELECTION, PROJECTION, COUNT, and COUNT DISTINCT all have a stability of 1; and GROUPBY has a stability of 2. Transformations with bounded stability constants propagate differential privacy guarantees made of their outputs back to their inputs, diminished by their stability constant.

Theorem 7.1. Let M provide ϵ-differential privacy, and let T be an arbitrary c-stable transformation. The composite computation $M \circ T$ provides $(\epsilon \times c)$-differential privacy.

Query Sensitivity	**PINQ:** $S(\cdot)$	FLEX: $\hat{S}^{(k)}(\cdot, x)$	PRIVATESQL: $\hat{\Delta}_{\mathtt{R}}(\cdot)$
$Count(r)$	$S(r)$	$\hat{S}^{(k)}_R(r, x)$	$\hat{\Delta}_{\mathtt{R}}(r)$
$\underset{G_1..G_n}{Count}(r)$	$2S(r)$	$2\hat{S}^{(k)}_R(r, x)$	$2\hat{\Delta}_{\mathtt{R}}(r)$

Relation Stability	**PINQ:** $S_R(\cdot)$	FLEX: $\hat{S}^{(k)}_R(\cdot, x)$	PRIVATESQL: $\hat{\Delta}_{\mathtt{R}}(\cdot)$
t	1	1	1 for $\mathtt{R}$; 0 for rest*
$r_1 \underset{a=b}{\bowtie} r_2$ General case	N.A.	$\mathsf{mf}_k(a, r_1, x)\hat{S}^{(k)}_R(r_2, x)+$ $\mathsf{mf}_k(b, r_2, x)\hat{S}^{(k)}_R(r_1, x)+$ $\hat{S}^{(k)}_R(r_1, x)\hat{S}^{(k)}_R(r_2, x)$	$\mathsf{mf}(a, r_1)\hat{\Delta}_{\mathtt{R}}(r_2)+$ $\mathsf{mf}(b, r_2)\hat{\Delta}_{\mathtt{R}}(r_1)+$ $\hat{\Delta}_{\mathtt{R}}(r_1)\hat{\Delta}_{\mathtt{R}}(r_2)$
No common ancestors		$\max(\mathsf{mf}_k(a, r_1, x)\hat{S}^{(k)}_R(r_2, x),$ $\mathsf{mf}_k(b, r_2, x)\hat{S}^{(k)}_R(r_1, x))$	$\max(\mathsf{mf}(a, r_1)\hat{\Delta}_{\mathtt{R}}(r_2),$ $\mathsf{mf}(b, r_2)\hat{\Delta}_{\mathtt{R}}(r_1))$
Special cases	$2S_R(r_1) \cdot S_R(r_2)^\star$		$\mathsf{mf}(b, r_2)\hat{\Delta}_{\mathtt{R}}(r_1) + \hat{\Delta}_{\mathtt{R}}(r_2)$, where $a \in \mathsf{keys}(r_1)^\star$
$\Pi_{a_1,\ldots,a_n} r$	$S_R(r)$	$\hat{S}^{(k)}_R(r, x)$	$\hat{\Delta}_{\mathtt{R}}(r)$
$\sigma_\varphi r$	$S_R(r)$	$\hat{S}^{(k)}_R(r, x)$	$\hat{\Delta}_{\mathtt{R}}(r)$
$\gamma_{a_1,\ldots,a_n} r$	$S_R(r)$	$\hat{S}^{(k)}_R(r, x)$	$\hat{\Delta}_{\mathtt{R}}(r)$
$\tau_{a,k} r$	N.A.	N.A.	$k\hat{\Delta}_{\mathtt{R}}(r)$

Table 7.2: Sensitivity and Stability. (*) PINQ supports a limited form of joins that truncates all except one matched record (described in §7.3). (*) PRIVATESQL tracks the key set of the relations and rewrites the query based on the privacy policy $\mathtt{R}$ (described in §7.5). Notation: $Count(r)$ counts the rows in relation r; $\underset{G_1..G_n}{Count}(r)$ groups rows in r by the specified groups $G_1, \ldots, G_n$; $r_1 \underset{a=b}{\bowtie} r_2$ joins r_1 and r_2 on $a = b$; $\Pi_{a_1,\ldots,a_n} r$ projects r on attributes $a_1, \ldots, a_n$; σ_φ r selects rows in r that satisfy φ; $\gamma_{a_1,\ldots,a_n}$ r groups r by attributes $a_1, \ldots, a_n$; $\tau_{a,k}$ r truncates rows in r that have a values appear more than k times; mf: maximum frequency.

Unlike other unary transformations, the JOIN operator takes two datasets as input with key selection functions for each, and returns the list of all pairs of elements whose keys match. It is possible that a single record can change arbitrarily number of output records, and thus unrestricted join operators have unbounded stability.

When to use PINQ. In PINQ, only a restricted form of JOIN is considered. This join requires that each input data set is first grouped by its join keys, and the list of groups are then joined using their group keys. The result is a compact representation of the output of the original JOIN, as each pair of groups could in principle be expanded to their full Cartesian product. This type of join has bounded stability, as each input record participates in at most one pair of groups, and as with GROUPBY the stability constant is at most 2. However, this restriction limits each join key result in a single record no matter how large the group is. Hence, you cannot extract more information privately. It is still better than leaving the stability unbounded and this join is useful to link unique identifiers between data sets.

PINQ has an open-sourced prototype, implemented as part of Language Integrated Queries (LINQ). Despite its limited support of join queries, the stability analysis has influenced the design of many systems or frameworks for DP.

7.4 Flex

The Flex system (Johnson *et al.*, 2018) uses an efficiently computed upper bound on local sensitivity, called *elastic sensitivity*, to bound the sensitivities of queries with general joins. Elastic sensitivity does not require modifications to the data, and it can be easily applied as a post-processing step to an existing join query. However, because elastic sensitivity is an upper bound on local sensitivity, a framework like smooth sensitivity (Nissim *et al.*, 2007) must be used to ensure that the sensitivity itself does not reveal too much about the data. The use of smooth sensitivity tends to increase noise, resulting in less accurate answers.

FLEX supports counting queries with grouping and equijoins, including self-joins and semijoins, and some kinds of subqueries. As described in Section 7.1, FLEX targets one-row bounded differential privacy. FLEX is limited to bounded differential privacy, because the smooth sensitivity framework is defined only for this setting.

The rules for calculating elastic sensitivity are given in Figure 7.2, in the "FLEX" column. In these rules, $\hat{S}^{(k)}(r, x)$ denotes the elastic sensitivity of the relation r at distance k from the actual database x, and $\hat{S}^{(k)}_R(r, x)$ denotes the stability of relation r at distance k from the actual database x. As in PINQ, the sensitivity of a counting query is exactly equal to the stability of the underlying relation, and the sensitivity of counting queries with grouping is doubled. As in PINQ, the stability of a base table is 1.

Maximum frequency. The stability of relational joins is defined in terms of $\mathsf{mf}_k(a, r, x)$, which denotes the *maximum frequency* of any single value for attribute a in relation r for actual database x. The *frequency* $freq(v)$ of the value $v \in dom(a)$ is the number of times attribute a takes the value v in the relation; the maximum frequency is $\max_{v \in dom(a)} freq(v)$. The mf_k function, defined in Figure 7.4, is an efficiently computed upper bound on maximum frequency for relational transformations. The base case for mf_k relies on the mf function, which is defined on base tables; $\mathsf{mf}(a, t, x)$ returns the *actual* maximum frequency for attribute a in table t of the database x. These values can be precomputed from the database x itself and stored as metadata.

mf_k	::	$a \to R \to D^n \to \mathbb{N}$
$\mathsf{mf}_k(a, t, x)$	$=$	$\mathsf{mf}(a, t, x) + k$
$\mathsf{mf}_k(a_1, r_1 \underset{a_2=a_3}{\bowtie} r_2, x)$	$=$	$\begin{cases} \mathsf{mf}_k(a_1, r_1, x)\mathsf{mf}_k(a_3, r_2, x) & a_1 \in r_1 \\ \mathsf{mf}_k(a_1, r_2, x)\mathsf{mf}_k(a_2, r_1, x) & a_1 \in r_2 \end{cases}$
$\mathsf{mf}_k(a, \Pi_{a_1,\ldots,a_n} r, x)$	$=$	$\mathsf{mf}_k(a, r, x)$
$\mathsf{mf}_k(a, \sigma_\varphi r, x)$	$=$	$\mathsf{mf}_k(a, r, x)$

Figure 7.4: Upper bound on maximum frequency from FLEX

Bounding stability. The intuition behind elastic sensitivity is as follows: for a join $r_1 \underset{a=b}{\bowtie} r_2$, each row in r_1 might match $\mathsf{mf}(b, r_2, x)$ rows in r_2 (in the worst case); symmetrically, each row in r_2 might match $\mathsf{mf}(a, r_1, x)$ rows in r_1 (in the worst case). We can therefore *increase* the number of rows in the result of the join by either $\mathsf{mf}(a, r_1, x)$ (by adding a row to r_2) or $\mathsf{mf}(b, r_2, x)$ (by adding a row to r_1). To produce a maximal increase, we take the maximum of these two. This approach works when there are no self-joins (the "no common ancestors" case in Figure 7.2).

For queries with self-joins, the increase might be even larger, because it is possible to add a row to *both* r_1 and r_2 simultaneously (e.g. if they are actually the same relation!). In this case, new matches from both relations could occur, and so we add them together (the "general case" case in Figure 7.2).

FLEX: applying elastic sensitivity. Since elastic sensitivity is computed using (exact) statistics from the data, scaling noise to elastic sensitivity directly might reveal more about the data than we intend. To address this issue, the smooth sensitivity framework (Nissim *et al.*, 2007) is used to ensure that the sensitivity of a query does not reveal too much about the data. The FLEX mechanism is defined as follows:

Definition 7.7 (The FLEX mechanism)**.** For query q on database x of size n, the following mechanism provides (ϵ, δ)-differential privacy:

1. Set $\beta = \frac{\epsilon}{2\log(2/\delta)}$
2. Calculate $S = \max_{k=0,\ldots,n} e^{-\beta k} \hat{S}^{(k)}(q, x)$
3. Release $q(x) + \mathsf{Lap}\left(\frac{2S}{\epsilon}\right)$

The FLEX mechanism is an instance of the smooth sensitivity framework with Laplace noise. Note that the elastic sensitivity is doubled in step (3)—this doubling of sensitivity is required by the smooth sensitivity framework, and can cause results to be less accurate than under global sensitivity-based approaches.

The approach requires a databases of fixed size (n), and is therefore limited to *bounded differential privacy*—a limitation inherited from

smooth sensitivity. In addition, the definition requires taking the maximum over all values of k between 0 and n, which can be computationally difficult if n is large. Fortunately, for the specific definition of elastic sensitivity, it can be shown that this maximum will always occur at small values of k: only values of k up to $\frac{j(q)^2}{\beta}$ need to be considered, where $j(q)$ denotes the number of syntactic joins in the query q.

When to use Flex. Flex is available as part of the open-source Chorus system (Johnson *et al.*, 2020a). Chorus is a framework for building systems for differential privacy, and requires more setup before deployment than systems like GoogleDP. However, Chorus works with any SQL database and supports a larger class of SQL-like queries than PINQ and GoolgeDP—in particular, queries that involve joins. Flex handles one query at a time and has a more efficient implementation of the Flex mechanism than the standard smooth sensitivity algorithm.

7.5 PrivateSQL

When the query workload is known in advance, the PrivateSQL system (Kotsogiannis *et al.*, 2019) will first generate a set of query views differentially privately, and then answer all the queries on top of these private views as a post-processing step. This allows more queries to be answered given a fixed privacy budget and also prevents timing attacks.

This section focuses on how PrivateSQL answers each view query V differentially privately and handles multi-relational databases with key constraints. First, compared to the sensitivity and stability rules of Flex in Figure 7.2 and the maximum frequency rules in Figure 7.5, PrivateSQL tracks keys in the query plan and hence allows much tighter upper bound for sensitivity of join queries. Second, PrivateSQL supports complex privacy policies where neighboring databases differ more than one row due to the constraints between the primary private table and the secondary private tables. Third, unlike Flex that applies local-sensitivity-based approach, PrivateSQL considers Lipschitz extension (Section 6.5) by transforming a given query Q to Q' by inserting truncation operators, such that the new query Q' has a bounded global sensitivity and also well approximates the original query answer.

Operations	$\mathsf{mf}(\mathbf{a}', s), \mathbf{a}' \subseteq attr(s)$	$\mathsf{keys}(s)$
$s = r_1 \underset{a_1=a_2}{\bowtie} r_2$	$\max(\mathsf{mf}(\bar{a}_2, r_1)\mathsf{mf}(a_2, r_2),$ $\mathsf{mf}(\bar{a}_1, r_2)\mathsf{mf}(a_1, r_1))$ where $\bar{a}_i = \mathbf{a}' - attr(r_i)$	$\{\mathbf{a}' \in \mathsf{keys}(r_2) \mid a_1 \in \mathsf{keys}(r_1)\} \cup$ $\{\mathbf{a}' \in \mathsf{keys}(r_1) \mid a_2 \in \mathsf{keys}(r_2)\}$
$s = \Pi_{a_1,\ldots,a_n} r$	$\mathsf{mf}(\mathbf{a}', r)$	$\{\mathbf{a}' \subseteq attr(s) \mid \mathbf{a}' \in \mathsf{keys}(r)\}$
$s = \sigma_{\varphi} r$	$\mathsf{mf}(\mathbf{a}', r)$	$\{\mathbf{a}' \subseteq attr(s) \mid \mathbf{a}' \in \mathsf{keys}(r)\}$
$s = \gamma_{a_1,\ldots,a_n} r$	$\mathsf{mf}(\mathbf{a}', r)$	$\{(a_1, \ldots, a_n)\} \cup$ $\{\mathbf{a}' \subseteq attr(s) \mid \mathbf{a}' \in \mathsf{keys}(r)\}$
$s = \gamma^{COUNT}_{a_1,\ldots,a_n} r$	$\mathsf{mf}(\mathbf{a}', r)$	$\{(a_1, \ldots, a_n)\} \cup$ $\{\mathbf{a}' \subseteq attr(s) \mid \mathbf{a}' \in \mathsf{keys}(r)\}$
$s = \tau_{a,k} r$	$\min(k, \mathsf{mf}(\mathbf{a}', r))$ if $a \subseteq \mathbf{a}'$; $\mathsf{mf}(\mathbf{a}', r)$, o.w.	$\{\mathbf{a}' \subseteq attr(s) \mid \mathbf{a}' \in \mathsf{keys}(r)\}$

Figure 7.5: Upper bound on maximum frequency and key set from PrivateSQL

We will start with simple privacy policy that neighboring databases only differ one row in the primary private table (e.g. Person Policy in Figure 7.2) to illustrate how PrivateSQL bounds sensitivity.

Bounding Sensitivity via Truncation Rewrite. The sensitivity bounds produced by the sensitivity calculator can be dependent on the max-frequency bounds on base relations. Adding *truncation operators* to the view query expression can bound the contributions of tuples from the base relations to the join output. These operators delete tuples that contain an attribute combination appearing in a join and whose frequency exceeds a *truncation threshold* k specified in the operator.

Definition 7.8 (Truncation Operator)**.** The truncation operator $\tau_{a,k}(r)$ takes in a relation r, a set of attributes a and a threshold k and for all $v \in dom(a)$, if the frequency of v in r is more than k, then any rows from r taking v for a is removed.

Truncation rewrite (see Algorithm 2) adds truncation operators to a view V and forms a new query plan V^{τ}. The algorithm takes as input a view V, a primary private relation R, and a vector of truncation thresholds $\mathbf{k}$, indexed by the attribute subset to which the threshold applies. It traverses every path p_l from relation r_l to the root operator and every join $r_1 \bowtie_{a_1=a_2} r_2$ on this path. If one of the join attributes is from R_l—say $a_1 \subseteq r_l$—and a_1 is not a key for r_1 and the primary

Algorithm 2: Truncation Rewrite $(V, \mathtt{R}, \mathbf{k})$

Initialize $V^\tau \leftarrow V$
for *every path* p_l *from leaf relation* r_l *to root in* V **do**
  for *every* $r_1 \bowtie_{a_1=a_2} r_2$ *on* p_l, *where* $a_1 \subseteq attr(r_l)$ **do**
    ▷(semijoin is also treated as a special equijoin)
    if $a_1 \notin Keys(r_1)$ *and* R *is a base relation of* r_2 **then**
      $k \leftarrow \mathbf{k}_{a_1}$
      Insert $\tau_{a_1,k}(r_l)$ above r_l in V^τ
      $\mathcal{A} \leftarrow \mathcal{A} \cup (a_1)$
return V^τ

private table R appears as a base relation in the expression r_2, then we insert $\tau_{a_1,k}(r_l)$ above r_l in V^τ.

After applying truncation rewrite to V, the estimated sensitivity no longer depends on the maximum frequency, but rather on the truncation thresholds. If the thresholds are set in a data independent manner, or learned using a differentially private mechanism, the sensitivity outputted by the calculator for V^τ is the global sensitivity of V^τ for simple policies. Note that this truncation operator introduces bias to the output and the choice of the truncation threshold affects the final query output.

For example, consider calculating the sensitivity of a view query:

SELECT relp, race, cnt FROM Person P, (SELECT COUNT(*) AS cnt, hid FROM PERSON GROUP BY hid) AS P2 WHERE P2.hid=P.hid

This query can be expressed using a relational algebra expression:

$$\pi_{race,relp,cnt}(Person \bowtie_{hid} (\gamma_{hid}^{COUNT}(Person))) \tag{7.2}$$

Based on the sensitivity rule of PrivateSQL from Table 7.2, this query would have a sensitivity bound of $(F \cdot 3 + 2)$, where F is the maximum frequency of hid in the Person relation. Now we insert a truncation operator before Person relation.

$$\pi_{race,relp,cnt}(\tau_{hid,k}(Person) \bowtie_{hid} (\gamma_{hid}^{COUNT}(Person))) \tag{7.3}$$

The truncation operator cuts down the maximum frequency of hid to k so that the sensitivity now becomes $3k$, even when the maximum frequency of the household id in Person relation is unbounded.

Handling Complex Policies via Semi-join Rewrite. Under complex privacy policies, neighboring databases differ in the primary private relation as well as other secondary private relations. For example, consider the Household privacy policy in Figure 7.2 — when a household row from the Household relation is removed, all the rows associated this this household will be removed. Computing the sensitivity of the query plans expressed in Eqns. (7.2) and (7.3) directly using the rules fail to give the true sensitivity of the query under the Household policy, as the Household relation never appears in this query plan. Hence, PrivateSQL introduces the *semijoin rewrite* that transforms view V into $V^{\ominus}$ such that the transformed view will give the correct sensitivity for complex privacy policies.

This rewrite works in two steps. First, it replaces every secondary private base relation r_j in V with a semijoin expression that makes explicit the transitive dependence between the primary private relation $\mathtt{R}$ and r_j. The resulting expression $V^{\ltimes}$ is such that $V(\mathbf{D}) = V^{\ltimes}(\mathbf{D})$. Moreover, the down sensitivity is now correct $\Delta_{\mathtt{R}}(V^{\ltimes}, \mathbf{D}) = \Delta^{\mathcal{C}}_{\mathtt{R}}(V^{\ltimes}, \mathbf{D})$ since transitive deletion is captured by the semijoin expressions. Second, to handle the high sensitivity of secondary private base relations, we add truncation operations using (Algorithm 2) to the semijoin expressions and transform $V^{\ltimes}$ to $V^{\ominus}$. Follow these two steps, the query plan expressed in Eqn. (7.2) will be transformed to

$$\pi_{race,relp,cnt}((\tau_{hid,k}(Person) \ltimes_{hid} Household) \bowtie_{hid} (\gamma^{COUNT}_{hid}(\tau_{hid,k}(Person) \ltimes_{hid} Household)) \tag{7.4}$$

This transformed expression allows the correctness of the query sensitivity calculation and also the correctness of the query answer.

Readers can refer to the full paper of PrivateSQL for the details of the sensitivity definition, and the semi-join rewrite, and the private learning of the truncation thresholds.

When to use PrivateSQL**.** If the true local sensitivity of a database instance is very large, then local sensitivity-based mechanism has to suffer large noise injection to the query answer, no matter how tight the upper bound is. For this case, truncation rewrites can help bring

down the noise if no many rows are truncated from the relations. PrivateSQL provides a flexible approach to handle privacy policies at different resolutions and algorithms for handling correlated subqueries. However, the current solutions only work for databases with foreign-key constraints. Extensions to general constraints are interesting open problems.

7.6 Google DP

Google DP (Wilson *et al.*, 2020) handles a special type of privacy policy for multi-relational databases with constraints: user-level DP. The approach is similar to the truncation used to handle joins in PrivateSQL. If a single user may contribute k records to the database, then the stability of a base table is actually k—not 1, as it is for all approaches listed in Table 7.1. Google DP assumes that a single user may contribute many records, and allows the analyst to specify a threshold k to bound the contribution of each user. The system applies a truncation rewrite to base tables that enforces the specified bound. The current implementation handles a large number of queries, but no correlated subqueries. Hence, it works well for a simpler schema that has a well defined "user" relation and simple SQL queries.

The same effect can be achieved in the other systems described in this Chapter by changing the stability of a base table (row t in Table 7.1) from 1 to k, and applying the truncation rewrite to base tables. The other rules for calculating stability and sensitivity remain the same.

Google DP also implements a novel strategy for GROUPBY. The challenge of GROUPBY is that the absence of a histogram bin in the results indicates (with certainty) a count of 0 for that bin. The other approaches in this Chapter solve this problem by requiring the analyst to specify the set of histogram bins. Google DP uses an approach called τ-thresholding (Korolova *et al.*, 2009) instead. The idea is to construct the set of histogram bins based on the data (which would normally violate differential privacy), but then to discard bins with counts below a threshold τ. This thresholding creates uncertainty about *why* a particular bin is missing—it could have had a count of 0, or it could have been below the threshold τ. The τ-thresholding approach

satisfies (ϵ, δ)-differential privacy.

The Google DP system has a well-maintained open-source implementation that acts as a standalone library or integrates directly with PostgreSQL.

8

Frameworks for Differentially Private Analysis

A wide range of frameworks have been developed by researchers and practitioners for differentially private analyses. They are useful for users with different levels of expertise in differential privacy, as shown in Figure 8.1. For experts in differential privacy, they can design a program from scratch for any applications or queries. However, they need to provide privacy proofs for their algorithms and show their accuracy guarantee for their interested applications or queries. Even experts can make non-trivial errors in this algorithm design process. For example, prior work by Lyu *et al.*, 2017 shows that pseudo-codes in research papers fail satisfying differential privacy as they claimed. Even the implementations of basic algorithms like the Laplace Mechanism and Exponential Mechanism can fail the desired privacy guarantee due to their use of floating point arithmetic (Mironov, 2012; Ilvento, 2020).

To address these challenges, programming frameworks including Ektelo (Zhang *et al.*, 2020), GoogleDP (Wilson *et al.*, 2020), SmartNoise (*Smart Noise* n.d.), IBM diffprivlib (*IBM Differential Privacy Library* n.d.), PrivInfer (Barthe *et al.*, 2016), LightDP (Zhang and Kifer, 2017) etc have been built to offer common packages and programs for differential privacy. Using these frameworks, programmers can directly

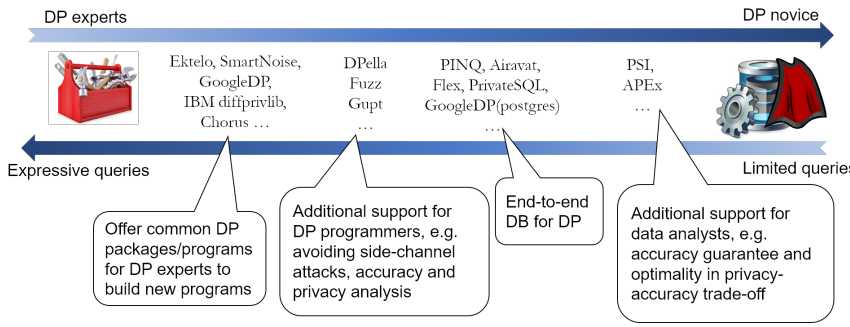

Figure 8.1: Frameworks for DP Analysis

use the state-of-the-art implementations to build new programs instead of starting from scratch. Besides the basic programming framework, there are additional features proposed in the literature to support DP programmers' programming experience. For example, Fuzz (Haeberlen *et al.*, 2011) raises the issue of side-channel attacks on programs that have a data-dependent running time and offers a general approach to prevent such attacks. Gupt (Mohan *et al.*, 2012) and DPella (Vesga *et al.*, 2020) offers an automatic accuracy analysis to the programmers. OpenDP project (*OpenDP* n.d.) provides a platform for trusted and open-source implementations of differentially private algorithms.

Chorus (Johnson *et al.*, 2020b) provides a framework for implementing differentially private algorithms that interoperate with existing SQL databases. Chorus does not propose a new approach or algorithm; instead, Chorus enables existing algorithms to scale more effectively to large datasets by interoperating with an existing high-performance database. For example, the sample & aggregate framework (Section 6.4) requires running the same query many times over subsets of the database. Chorus provides a query rewriting module to allow the transformation of an analyst-specified query into a new one that implements the necessary execution over subsets of the data. This idea has since been adopted in similar modules in Google DP (Wilson *et al.*, 2020) and SmartNoise (*Smart Noise* n.d.) to interoperate with scalable databases.

How about users who do not know how to write a differentially private programs? We have seen in Chapter 7 that frameworks like PINQ

(McSherry, 2009), Flex (Johnson *et al.*, 2018), and PrivateSQL (Kotsogiannis *et al.*, 2019) offer SQL-like query interface for data analysts to access the private data. With these end-to-end database systems, the data analysts are not required to design or write a differentially private program, but simply write a SQL-like query with a privacy budget. The system itself ensures that a differentially private mechanism is run with the privacy budget specified by the data analyst. GoogleDP for PostgreSQL (Wilson *et al.*, 2020) and SmartNoise also support similar framework and Airavat (Roy *et al.*, 2010) builds this end-to-end framework on top of a MapReduce system. However, these frameworks do not design for utility-optimal programs. Many state-of-the-art algorithms have not been implemented or considered in them. Moreover, the data analyst usually does not know how to set the privacy budget for each of their queries. This motivated another class of accuracy-aware frameworks for non-experts in DP, such as APEx (Ge *et al.*, 2019) and PSI (Murtagh *et al.*, 2018). In particular, APEx allows the data analyst to specify a query with an accuracy requirement for the query. On behalf of the data analyst, APEx picks the best algorithm that consumes the least privacy budget and returns an answer that satisfies the accuracy requirement of the data analyst.

Next, we will use ϵktelo to illustrate how a DP expert can effectively write a DP program (Section 8.1) and APEx to illustrate how a non-expert can effectively query a DB system (Section 8.2).

8.1 ϵktelo

ϵktelo is a programming framework and system that aids programmers in developing programs with differential privacy guarantee and high utility. This program supports a large class of statistical tasks such as releasing contingency tables, multi-dimensional histograms, answering OLAP and range queries, and even implementing private machine learning algorithms. In ϵktelo, differentially private algorithms are described using plans composed over a rich library of operators. Most of the plans are linear sequence of operators, but ϵktelo also supports plans with iteration, recursion, and branching. Five *operator classes* are carefully chosen based on their input-output specifications: (a) transformation,

(b) query, (c) inference, (d) query selection, and (e) partition selection. Each opeator belongs to one of these five classes.

Algorithm 3: ϵktelo CDF Estimator

1: $D \leftarrow$ Protected(source_uri) ▷ Init
2: $D \leftarrow$ Where(D, sex ==‘M’ AND age $\in [30, 39]$ ▷ Transform
3: $D \leftarrow$ Select(salary) ▷ Transform
4: $\mathbf{x} \leftarrow$ T-Vectorize(D) ▷ Transform
5: $\mathbf{P} \leftarrow$ AHPPartition($\mathbf{x}, \epsilon/2$) ▷ Partition Select
6: $\bar{\mathbf{x}} \leftarrow$ V-ReducedByPartition($\mathbf{x}, \mathbf{P}$) ▷ Transform
7: $\mathbf{M} \leftarrow$Identity($|\bar{\mathbf{x}}|$) ▷ Query Select
8: $\mathbf{y} \leftarrow$ VecLaplace($\bar{\mathbf{x}}, \mathbf{M}, \epsilon/2$) ▷ Query
9: $\hat{\mathbf{x}} \leftarrow$ NNLS($\mathbf{P}, \mathbf{y}$) ▷ Inference
10: $\mathbf{W}_{pre} \leftarrow$ Prefix($|\mathbf{x}|$) ▷ Query Select
11: **return** $\mathbf{W}_{pre} \cdot \hat{\mathbf{x}}$ ▷ output

We use Algorithm 3, a ϵktelo plan from the full paper to illustrate these operators. This plan takes as input a table D with schema of three attributes Age, Gender, and Salary and estimate the empirical cumulative distribution function (CDF) of the Salary attribute for males in their 30’s. The plan first applies transformation operators to select rows that satisfy the conditions (Line 2) and then project only the Salary attribute (line 4). Then it applies another transformation operator to construct a vector of counts $\mathbf{x}$ for, where entry $\mathbf{x}[i]$ represent the number of rows for each salary value i.

Then the plan spends $\epsilon/2$ privacy budget to partition the data domain using a *partition selection* operator (Line 5). The resulted partition $\mathbf{P}$ is used to transform the data vector $\mathbf{x}$ into a new vector of counts $\bar{\mathbf{x}}$ over the partitions. Then an identity query is selected (Line 7) and Laplace mechanism is applied to the identity query over $\bar{\mathbf{x}}$ with the remaining privacy budget $\epsilon/2$ (Line 8). The sensitivity of the query $\mathbf{M}$ will be automatically computed by ϵktelo.

Given the noisy counts $\mathbf{y}$ over the partition $\mathbf{P}$, the plan applies *inference* operator NNLS (short for non-negative least squares) to infer non-negative counts in the original vector space of $\mathbf{x}$ (Line 9). Finally, the CDF query is selected (Line 10) and applied to the inferred noisy counts $\hat{\mathbf{x}}$ (line 11).

This framework demonstrates four important design principles:

- Expressiveness: a large set of state-of-the-art differentially private algorithms on tabular data can be written as ϵktelo plans.
- Privacy "for free": any plan written in ϵktelo automatically satisfy differential privacy.
- Reduced privacy verification effort: the implementation of individual operators in ϵktelo needs to be vetted only once and can be reused in different plans.
- Transparency: comparisons of different differentially private programs in terms of ϵktelo plans are intuitive and easy.

When to use ϵktelo. ϵktelo is the first DP programming framework with predefined python packages for complex algorithms that handle counting-based queries. The same set of design principles in the frameworks or toolboxes are also offered in new DP tools developed by the industry, including GoogleDP, IBM diffprivlib, SmartNoise, OpenDP, Chorus, etc. These tools include DP primitives such as Laplace mechanism, Guassian mechanism, and packages for advanced privacy composition algorithm (Section 2.4.2) in different programming languages. This makes differential privacy programming more accessible to researchers and developers.

8.2 APEx

The general purpose differentially private query answering systems (e.g. PINQ McSherry, 2009) allow users to write differentially private programs and ensure that every program expressed satisfies differential privacy. However, to achieve high accuracy, the analyst typically would need to know the optimal methods from the privacy literature on how to add noise to answers. These systems do not provide any guarantees to the data analyst on the quantity they really care about, namely *accuracy* of query answers. In fact, most of these systems take a privacy level (ϵ) as input and make sure that differential privacy holds, but leave accuracy unconstrained.

APEx proposed by Ge *et al.*, 2019 allows data analysts to explore a sensitive dataset D held by a data owner by posing a sequence of declaratively specified queries that can capture typical data exploration workflows. The system aims at achieving a dual goal: (1) since the data are sensitive, the data owner would like the system to provably bound the information disclosed about any one record in D to the analyst; and (2) since privacy preserving mechanisms introduce error, the data analyst must be able to specify accuracy bounds on each query.

To allow the data analyst to explore data with bounded error, APEx considers three classes of exploration queries and extend them to incorporate an accuracy requirement. The syntax for accuracy is inspired by that in BlinkDB Agarwal *et al.*, 2013:

BIN D ON $f(\cdot)$ WHERE $W = \{\phi_1, \ldots, \phi_L\}$
[HAVING $f(\cdot) > c$]
[ORDER BY $f(\cdot)$ LIMIT k]
ERROR α CONFIDENCE $1 - \beta$;

Each query is associated with a *workload* of predicates $W = \{\phi_1, \ldots, \phi_L\}$. Based on W, the tuples in a table D are divided into bins. Each bin b_i contains all the tuples in D that satisfy the corresponding predicate $\phi_i : dom(R) \rightarrow \{0, 1\}$, i.e., $b_i = \{r \in D | \phi(r) = 1\}$. More a query has an aggregation function $f : dom(R)^* \rightarrow \mathbb{R}$, which returns a numeric answer $f(b_i)$ for each bin b_i. The output of this query without the optional clauses (in square brackets) is a list of counts $f(b_i)$ for bin b_i. We denote this class of query workload counting queries (WCQs).

Each query can be specialized using one of two optional clauses: The HAVING clause returns a list of bin identifiers b_i for which $f(b_i) > c$; and the ORDER BY ... LIMIT clause returns the k bins that have the largest values for $f(b_i)$. Throughout this paper, we assume $f(\cdot)$ is the COUNT function and omit extensions to other aggregates like AVG, SUM, QUANTILE due to space constraints. Queries with the Having clause returns predicates that have counts more than the threshold are called iceberg counting queries (ICQs). Queries with the ORDER BY ... LIMIT are called top-k counting queries (TCQs).

Each query has its own accuracy specification. For example, the accuracy requirement for a WCQ q_W is defined as a bound on the maximum error across queries in the workload W.

Definition 8.1 ((α, β)-WCQ accuracy)**.** Given a workload counting query $q_W : \mathcal{D} \to \mathbb{R}^L$, where $W = \{\phi_1, \ldots, \phi_L\}$. Let $M : \mathcal{D} \to \mathbb{R}^L$ be a mechanism that outputs a vector of answers y on D. Then, M satisfies (α, β)-W accuracy, if $\forall D \in \mathcal{D}$,

$$\Pr[\|y - q_W(D)\|_\infty \geq \alpha] \leq \beta, \tag{8.1}$$

where $\|y - q_W(D)\|_\infty = \max_j |y[i] - c_{\phi_i}(D)|$.

APEx has two components. The first component *accuracy translator* takes in an analyst query (q, α, β) and chooses a mechanism M that can (a) answer q under the specified accuracy bounds, with (b) minimal privacy loss. To achieve both these desiderata, APEx supports a set of differentially private mechanisms that can be used to answer each query type. Multiple mechanisms are supported for each query type as different mechanisms result in the least privacy loss depending on the query and the dataset.

Each mechanism M exposes two functions: M.TRANSLATE, which *translates* a query and accuracy requirement into a lower and upper bound (ϵ^l, ϵ^u) on the privacy loss if M is executed, and M.RUN that runs the differentially private algorithm and returns an approximate answer ω for the query. ω is guaranteed to satisfy the specified accuracy requirement. Moreover, M satisfies ϵ^u differential privacy.

We first introduce one basic translation mechanism in APEx shown in Algorithm 4. This mechanism is based Laplace mechanism. The constant $\|\mathbf{W}\|_1$ is equal to the sensitivity of queries set defined by the workload $\mathbf{W}$. It measures the maximum difference in the answers to the queries in $\mathbf{W}$ on any two databases that differ only a single record and is equal to the maximum absolute column sum of $\mathbf{W}$.

Algorithm 4 provides the RUN and TRANSLATE of Laplace mechanism for all three query types. This algorithm first transforms the query q_W and the data D into matrix representation $\mathbf{W}$ and $\mathbf{x}$. The TRANSLATE outputs a lower and upper bound (ϵ^l, ϵ^u) for each query type with a given accuracy requirement and these two bounds are the same as

Algorithm 4: Laplace Mechanism (LM) (q, α, β, D)

$\mathbf{W} \leftarrow \mathcal{T}(W = \{\phi_1, \ldots, \phi_L\}), \mathbf{x} \leftarrow \mathcal{T}_W(D), \alpha, \beta$

Function RUN(q, α, β, D)**:**
- $\epsilon \leftarrow$ TRANSLATE$(q_W, \alpha, \beta).\epsilon^u$
- $[\tilde{x}_1, \ldots, \tilde{x}_L] \leftarrow \mathbf{Wx} + Lap(b)^L$, where $b = \|\mathbf{W}\|_1/\epsilon$
- **if** *q.type==WCQ (i.e., q_W)* **then**
  - **return** $([\tilde{x}_1, \ldots, \tilde{x}_L], \epsilon)$
- **else if** *q.type==ICQ (i.e., $q_{W,>c}$)* **then**
  - **return** $(\{\phi_i \in W \mid \tilde{x}_i > c\}, \epsilon)$
- **else if** *q.type==TCQ (i.e., $q_{W,k}$)* **then**
  - **return** $(\texttt{argmax}^k_{\phi_1,\ldots,\phi_L} \tilde{x}_i, \epsilon)$

Function TRANSLATE(q, α, β)**:**
- **if** *q.type==WCQ (i.e., q_W)* **then**
  - **return** $(\epsilon^u = \frac{\|\mathbf{W}\|_1 \ln(1/(1-(1-\beta)^{1/L}))}{\alpha}, \epsilon^l = \epsilon^u)$
- **else if** *q.type==ICQ (i.e., $q_{W,>c}$)* **then**
  - **return** $(\epsilon^u = \frac{\|\mathbf{W}\|_1 (\ln(1/(1-(1-\beta)^{1/L})) - \ln 2)}{\alpha}, \epsilon^l = \epsilon^u)$
- **else if** *q.type==TCQ (i.e., $q_{W,k}$)* **then**
  - **return** $(\epsilon^u = \frac{\|\mathbf{W}\|_1 2(\ln(L/(2\beta)))}{\alpha}, \epsilon^l = \epsilon^u)$

Laplace mechanism is data independent. However, these bounds vary among query types. The RUN takes the privacy budget computed by TRANSLATE(q, α, β) (Line 3) and adds the corresponding Laplace noise $[\tilde{x}_1, \ldots, \tilde{x}_L]$ to the true workload counts $\mathbf{Wx}$. When q is a WCQ, the noisy counts are returned directly; when q is an ICQ, the bin ids (the predicates) that have noisy counts $\geq c$ are returned; when q is a TCQ, the bin ids (the predicates) that have the largest k noisy counts are returned. Beside the noisy output, the privacy budget consumed by this mechanism is returned as well. The following theorem summarizes the properties of the two functions RUN and TRANSLATE.

Theorem 8.1. Given a query q where q.type $\in \{WCQ, ICQ, TCQ\}$, Laplace mechanism (Algorithm 4) denoted by M can achieve (α, β)-q.type accuracy by executing the function RUN(q, α, β, D) for any $D \in \mathcal{D}$,

and satisfy differential privacy with a minimal cost of TRANSLATE(q, α, β) $.\epsilon^u$.

Note that the privacy translation for Laplace mechanism is not data dependent, i.e., $\epsilon^l = \epsilon^u$. Next, we will illustrate a data dependent translation algorithm with a special case of ICQ when the workload size $L = 1$, denoted by $q_{\phi,>c}(\cdot)$. Intuitively, when the true count $c_\phi(D)$ is much larger (or smaller) than the threshold c, then a much larger (smaller resp.) noise can be added to $c_\phi(D)$ without changing the output of the system.

Example 8.1. Consider a query $q_{\phi,>c}$, where $c = 100$. To achieve (α, β) accuracy for this query, where $\alpha = 10, \beta = 0.1^{10}$, the Laplace mechanism requires a privacy cost of $\frac{\ln(1/(2\beta))}{\alpha} = 2.23$ by Theorem 8.1, regardless of input D. Suppose $c_\phi(D) = 1000$. In this case, $c_\phi(D)$ is much larger than the threshold c, and the difference is $\frac{(1000-100)}{\alpha} = 90$ times of the accuracy bound $\alpha = 10$. Hence, even when applying Laplace comparison mechanism with a privacy cost equals to $\frac{2.23}{90} \approx 0.25$ wherein the noise added is bounded by 90α with high probability $1-\beta$, the noisy difference $c_\phi(D) - c + \eta_{sign}$ will still be greater than 0 with high probability.

This example shows that a different mechanism can result in the same accuracy guarantee with a smaller privacy cost compared to Laplace mechanism. However, the tightening of the privacy cost in the example above requires to know the true count value. To tackle this challenge, APEx applies a Multi-Poking Mechanism. This mechanism involves multiple pokes with improving accuracy requirements (increasing privacy cost resp.), while each poke checks if bins have either sufficiently large noisy differences with respect to the accuracy requirement. This allows the data analyst to learn the query answer with a gradual relaxation of privacy cost. The privacy loss incurred by such a mechanism may be $\epsilon \in (\epsilon^l, \epsilon^u)$ that is smaller than the worst case, depending on the characteristics of the dataset. Readers can refer to the full paper for the details.

Hence, the second component of APEx is *privacy engine* that ensures the privacy budget B specified by the data owner is not violated. Given a sequence of queries $(M_1, \ldots, M_i)$ already executed by the privacy

engine that satisfy an overall B_{i-1}-differential privacy and a new query (q_i, α_i, β_i), APEx identifies a set of mechanisms $\mathcal{M}^*$ that all will have a worst case privacy loss smaller than $B - B_{i-1}$. That is, running any mechanism in $\mathcal{M}^*$ will not result in exceeding the privacy budget in the worst case. If $\mathcal{M}^* = \emptyset$, then APEx returns 'Query Denied' to the analyst. Otherwise, APEx runs one of the mechanisms M_i from $\mathcal{M}^*$ by executing $M_i.\text{RUN}()$ and the output ω_i will be returned to the analyst. APEx then increments B_{i-1} by the actual privacy loss ϵ_i rather than the upperbound ϵ^u. As explained above, in some cases $\epsilon_i < \epsilon^u$ as different execution paths in the mechanism can have different privacy loss. Nevertheless, the privacy analyzer guarantees that the execution of any sequence of mechanisms $(M_1, M_2, \ldots, M_i)$ before it halts is B-differentially private.

When to use APEx. The accuracy-aware feature proposed by APEx is important for data analysts to understand the accuracy guarantee of their query answers without being a DP expert. The current implementation of APEx involves only a single table, and three classes of queries for private data exploration. The database systems than handle more general queries such as PrivateSQL and Flex, only include one DP algorithm for each query, and this algorithm may not necessarily result in the minimum privacy loss when fixing the accuracy requirement. Hence, it is recommended to integrate more algorithms and accuracy-aware features into these end-to-end database systems for DP.

9

Eliminating the Trusted Data Curator

The techniques we have seen so far target the *central model* of differential privacy, in which data is collected centrally by a *trusted data curator* who runs mechanisms on behalf of the analyst. The central model is unrealistic in many contexts, because the data curator may *not* be trustworthy.

Such assumptions about who can be trusted are encoded in a *threat model*, introduced in Chapter 2. The threat model describes who the adversary is and what their capabilities are. In the central model of differential privacy, we assume that the adversary is *not* capable of corrupting the data curator.

A significant amount of research investigates approaches that eliminate the need for a trusted data curator, and thus result in a threat model that makes fewer trust assumptions. The *local model* of differential privacy is the oldest of these (in fact, randomized response predates the development of differential privacy itself). More recent developments include the *shuffle model*, in which groups of participants have their responses shuffled for anonymity by a designated *shuffler* to amplify the privacy guarantee of the local model, and approaches that leverage *secure computation* to *simulate* a trusted data curator when the data curator may actually be malicious.

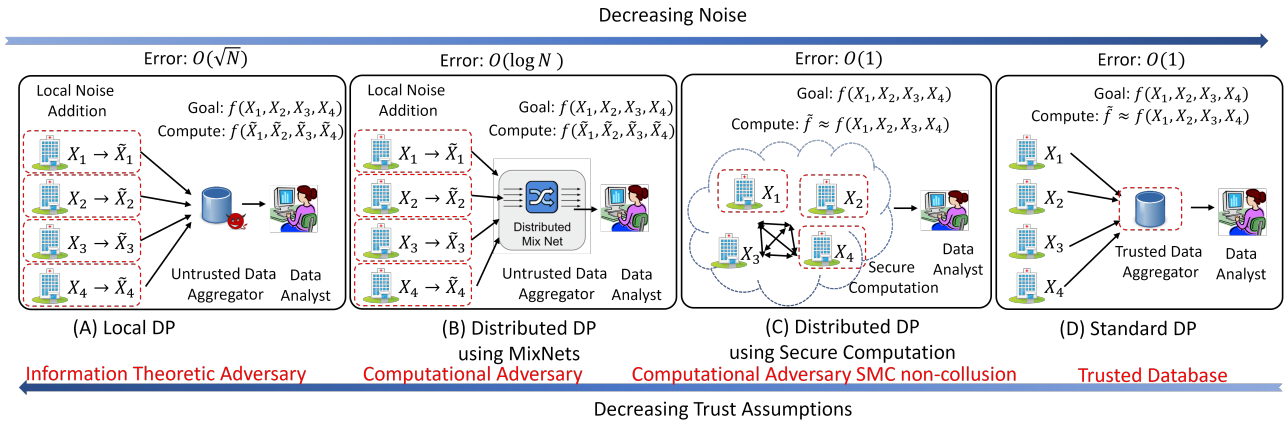

Figure 9.1: Architectures of four models for differential privacy. In the *central model*, the data curator is trusted; the other three models reduce the trust required in the data curator.

Each model results in slightly different trust assumptions, encoded in its own threat model. If these assumptions hold, then mechanisms in the model satisfy differential privacy. If they do not, then there is no guarantee. Figure 9.1 summarizes the architecture differences between the four models. In this chapter, we provide an overview of the major known techniques in each of these models. We describe mechanisms for the local model in Section 9.1, for the shuffle model in Section 9.2, and for secure computation in Section 9.3.

9.1 The Local Model

In the local model of differential privacy, participants add noise to their data *before* submitting it to the data curator. In this setting, the data curator holds *only* differentially private data—so the data curator does not need to be trusted. The local model has obvious and significant advantages over the central model: the participants' data is protected by differential privacy, even if the data curator is untrustworthy or the central server holding the data is compromised, and no further security precautions are needed to obtain this guarantee. The threat model in the local model of differential privacy requires very few trust assumptions—it allows the adversary to control the data curator and all data owners except one, and still provides differential privacy.

The primary disadvantage of the local model is that it requires a *lot* of noise compared to the central model. For example, the absolute error introduced by the Laplace mechanism is $O(1)$; the absolute error of mechanisms in the local model is at least $O(\sqrt{n})$, where n is the number of participants. For a query whose "signal" grows with $O(n)$, local differential privacy mechanisms can still provide useful results—but often require significantly more participants than central-model approaches to do so.

9.1.1 Randomized Response

The prototypical mechanism for local differential privacy is *randomized response* (Warner, 1965), which predates the development of differential privacy by decades. Randomized response allows the data curator to

collect differentially private responses to a single "yes/no" question, and was originally developed to aid in collecting survey data about sensitive topics. A brief description of randomized response was given in Chapter 2; the full protocol for collecting a single response is as follows:

1. The responder flips a coin. If the result is heads, the responder answers the question truthfully.

2. Otherwise, the responder flips a second coin. If the second coin's result is heads, the responder answers "yes." Otherwise, the responder answers "no."

The privacy in this protocol comes from the second case—in which the responder answers randomly—and the fact that the interviewer does not know which case was chosen.

9.1.2 Histogram Queries

Randomized response is suitable for answering a single counting query, but not for more complicated queries. A number of different mechanisms have been developed to answer histogram queries; many of these can be thought of as instances of framework with three steps: *encode*, *perturb*, and *aggregate* (Wang *et al.*, 2017). One example is *unary encoding*, which works for a histogram query over a domain d of options. The approach has two parameters, p and q, that control the strength of the privacy guarantee.

1. **Encode:** For a response $v \in d$, generate a *one-hot* encoding B of v as a length-$|d|$ binary vector with all elements 0 except for the one corresponding to v.

2. **Perturb:** Generate the perturbed encoding B' by flipping bits of B according to the following probability distribution:

$$\mathsf{Pr}[B'[i] = 1] = \begin{cases} p & \text{if } B[i] = 1 \\ q & \text{if } B[i] = 0 \end{cases}$$

3. **Aggregate:** After collecting n responses, generate the vector of approximate counts A for each element of d as follows:

$$A[i] = \frac{\sum_j B'_j[i] - nq}{p - q}$$

Theorem 9.1 (Privacy for Unary Encoding (Wang *et al.*, 2017))**.** For parameters p and q, the unary encoding approach satisfies ϵ-local differential privacy for:

$$\epsilon = \log\left(\frac{p(1-q)}{(1-p)q}\right)$$

The unary encoding approach is similar to the solution of RAPPOR (Erlingsson *et al.*, 2014). The same authors formalize several more complex techniques that yield improvements in accuracy (Wang *et al.*, 2017), but all of them result in $O(\sqrt{n})$ error.

9.1.3 More Complex Queries

Due to the challenges of the model, the majority the work in local differential privacy has focused on single queries or specific sets of queries (like histograms).

Recent work has resulted in more general mechanisms for releasing higher-dimensional results with local differential privacy. The CALM technique (Zhang *et al.*, 2018) releases differentially private marginals in the local model; the HIO (Wang *et al.*, 2019) approach extends this idea to analytical queries over many attributes, and HDG (Yang *et al.*, 2020) further extends the approach to range queries.

To answer workloads of linear queries, a *workload factorization mechanism* (McKenna *et al.*, 2020) can be used. This approach extends ideas from the matrix mechanism in the central model (described in Section 4.2) to the local model, and is capable of generating optimized mechanisms for answering specific workloads. The authors have shown that existing mechanisms in the local model can be viewed as specific strategies described using their workload factorization approach.

9.2 The Shuffle Model

The *shuffle model* (Bittau *et al.*, 2017; Erlingsson *et al.*, 2019) is a compromise between the local and central models that retains some of the benefits of both. The shuffle model works by using several *shufflers* to ensure anonymity of data owners before data reaches the data curator.

Anonymous communication channels (Chaum, 1981) enable a user to remain unidentifiable from a set of other users (called the anonymity set). Examples of such systems include Mixnets, which use proxies to mix communications from various users. The key insight of the shuffle model is that anonymous communication results in a kind of *privacy amplification* (Kasiviswanathan *et al.*, 2011; Balle *et al.*, 2018).

In the shuffle model, data owners submit randomized data to a *shuffler* (i.e. data owners add noise to their data *before* it leaves their device—just as in local differential privacy). This randomization process is similar to the encode and perturb steps in Section 9.1 for local model. Once it has received data from at least n data owners, the shuffler "anonymizes" the batch of n data records by removing identifying information and metadata and shuffling the records. Then, the shuffler submits the entire batch of n records to the data curator. This process makes it impossible for the data curator to link any single record to its owner, except using the data contained in the record itself (i.e. it creates an anonymous communication channel). The records will be aggregated in a similar way as the aggregate step in Section 9.1 for local model.

Like local differential privacy, the shuffle model derives its privacy guarantee by asking data owners themselves to add noise to their data. However, the anonymous communication channel amplifies the resulting privacy guarantee—so data owners can add *less noise* to their data before submitting it. As first implemented in the Prochlo system (Bittau *et al.*, 2017; Erlingsson *et al.*, 2019), the approach outlined above reduces error from $O(\sqrt{n})$ (in LDP) to $O(\log(n))$.

Threat model. The shuffle model requires slightly different trust assumptions than the local model. As in local differential privacy, the data curator does not need to be trusted. However, the newly-introduced shuffler *must* be trusted—if the adversary corrupts the shuffler and

breaks the intended anonymity property, then no privacy amplification is achieved by the shuffling process.

The need to trust the shuffler can be addressed in several ways. First, *multiple* shufflers can be used, which effectively distributes trust among them. To be sure of intercepting a target record, an adversary would need to corrupt *all* of the shufflers. Second, other techniques for secure computation can be used to ensure that shufflers are not corrupted; for example, Prochlo implements the shuffling step using *trusted hardware* to eliminate the need for a trusted shuffler.

Further reducing error. The shuffle model of differential privacy represents an active area of research. Recent work has clarified the limits of the shuffle model and brought it closer to the central model in terms of accuracy. For example, we now know that there are some mechanisms in the central model that do not have shuffle-model analogues with the same error. In particular, anonymous communication with a single message per data owner cannot yield expected error less than $O(N^{1/6})$ (Balle *et al.*, 2019). On the other hand, for some types of analysis, the shuffle model produces the *same error* as the central model: with a constant number of messages per data owner, it is possible to reduce the error for real-valued DP summation in the shuffle model to $O(1)$ (Ghazi *et al.*, 2020; Balle *et al.*, 2020).

9.3 Leveraging Secure Computation

A third approach for reducing trust assumptions while maintaining the low error of the central model is to employ techniques from cryptography for *secure computation*. Specifically, *homomorphic encryption* (HE) and *secure multiparty computation* (MPC) techniques can eliminate the central model's need for a trusted data curator (Dwork *et al.*, 2006a; Narayan and Haeberlen, 2012; Bater *et al.*, 2017; Agarwal *et al.*, 2018; Roth *et al.*, 2019). Generally speaking, these techniques allow a group of participants to compute some function of shared data, revealing only the function's output—even in the absence of a trusted third party.

Secure computation techniques are naturally complementary to differential privacy. Secure computation ensures that all parties learn

only the output of the computation (and nothing else), while differential privacy bounds the information leakage of the output itself. Secure computation controls *who gets to learn* each input and output (a security property), while differential privacy controls *what can be learned* from that output (a privacy property).

Combining the two allows us to *simulate* the trusted data curator required in the central model, even if no trusted data curator is available. Secure computation techniques therefore directly enable the use of low-error central model mechanisms (i.e. $O(1)$ error) while relaxing the trust assumptions of the central model.

Example: DJoin. Consider a simple setting where two parties would like to compute the intersection size of their data while preserving differential privacy for both datasets. If each party does not trust each other, how can we ensure a constant additive error as if they trust each other? It is well known that the lower bound for this query is $\sqrt{N}$, where N is the data size of each party (McGregor *et al.*, 2010), if we want to ensure the view of each party satisfies differential privacy. However, if we assume both parties are computationally bounded, a constant additive error can be achieved.

DJoin (Narayan and Haeberlen, 2012) offers a concrete protocol for achieving differential privacy under this assumption. This protocol applies private set-intersection cardinality technique to privately compute the noisy intersection set of the two datasets. First, party A defines a polynomial over a finite field whose roots are the elements owned by A. Party A then sends the homomorphic encryptions of the coefficients to party B, along with its public key. Then the encrypted polynomial is evaluated at each of Party B's inputs, followed by a multiplication with a fresh random number. The number of zeros in the results is the true intersection size between A and B. To provide differential privacy, party B adds a number of zeros (differentially-private noise of $O(1)$ independent of data size) to the results and sends the randomly permuted results back to party A. Party A decrypts the results and counts the number of zeros. Party A also adds another copy of differentially private noise to the count and sends the result it back to party B. In other words, both parties add noise to their inputs to achieve privacy.

However, the final protocol output has only an error of $O(1)$, which is the same as the corresponding central model approach.

Limitations & open problems. Using secure computation and encryption achieves the same error as the central model and prevents any party from seeing the other party's input in the clear. However, this requires an additional assumption of all parties being computationally bounded in the protocol. Hence, the type of differential privacy guarantee achieved in DJoin is known as *computational differential privacy* (Mironov *et al.*, 2009). In addition, most of the existing protocols consider honest-but-curious adversaries who follow the protocol specification or consider malicious adversaries with an additional overhead to enforce honest behaviour i.e., verify that the computation was performed correctly.

10

Implementation Issues & Open Challenges

We conclude with a survey of issues specific to implementations of differentially private algorithms, and a discussion of open challenges *not* covered in earlier in earlier Chapters. Differential privacy has already enjoyed significant success; many of the open challenges in differential privacy are related to developing techniques for new kinds of data (*e.g.* unstructured or streaming data), setting the privacy budget, communicating with non-experts about the privacy guarantee, and supporting analysts in applying differentially private algorithms.

10.1 Privacy Definitions & Algorithm Design

The work we have covered in this book is largely focused on the setting of tabular data stored in a database, in which each individual contributes a single row of data. The definition of neighboring databases given in Chapter 2 formalizes this notion, and that definition is common in research on differential privacy.

The definition of differential privacy also applies in other contexts, but each of these requires a new definition of neighboring databases. Finding a *good* formal distance metric for databases can be surprisingly challenging; the definition should reflect the informal notion that neigh-

boring databases differ in one person's data—which can be very difficult to define formally. This section briefly summarizes the challenges associated with particular forms of data and some of the work that has been done in these areas.

Multi-relational data. Chapter 7 describes the challenges of answering queries over multi-relational databases with differential privacy. This setting presents two major challenges: the high or unbounded sensitivity of the queries involved, and the ability to express *policies* that constrain the structure of the database itself. Most of the existing research has focused on the first challenge; PrivateSQL (Kotsogiannis *et al.*, 2019) is the only approach we cover that addresses the second.

The question of *which databases are neighbors* is at the core of this second challenge, and answering this question definitively remains a serious challenge (as demonstrated by the example at the beginning of Chapter 7). An incorrect answer to this question may result in insufficient privacy protection, even though the formal privacy property holds.

Growing Databases & Streaming Data. The vast majority of the work in differential privacy for databases has assumed that the data is fixed and does not change over time. In practice, however, databases *do* change over time—often continuously. This presents both challenges and opportunities for differential privacy, many of which remain unexplored.

The privacy parameter ϵ is often considered a global privacy "budget," but in the growing database setting, a single global budget is a pessimistic over-estimate of the privacy risk to some participants—in particular, newly added data has never been queried, so its owners are not subject to any privacy risk at all! Several approaches aim to take advantage of this fact by tracking privacy budget with finer granularity as data is added to the database (*e.g.* Ebadi *et al.*, 2015; Cummings *et al.*, 2018). Such approaches may prove necessary to address the common concern that in practical deployments of differential privacy for database queries, the privacy budget will be quickly exhausted.

Streaming data presents similar challenges to growing databases; in addition, streaming data often involves multiple data contributions

by a single user. A number of approaches have been developed for releasing differentially private statistics about streaming data (*e.g.* Chan *et al.*, 2010; Chen *et al.*, 2017). The definition of neighboring databases (or *neighboring streams*, in this setting) is a key part of the problem definition for this area. Many approaches consider a kind of *sliding-window* for privacy: they assume that two neighboring streams will differ only in a bounded number of events within a particular window, and will be identical otherwise. Whether or not this kind of definition provides sufficient privacy for streaming data in practice is an open question. A stricter definition—for example, one in which neighboring streams may differ arbitrarily in one individual's contributions—tends to result in statistics with too much noise to be usable.

Correlated Data. The implicit assumption of the "standard" definition of neighboring databases is that rows are independent—knowing that row A is present in the database does not imply new knowledge about whether or not row B is present. Violating this assumption does not "break" differential privacy—the mathematical property still holds, either way—but it might result in surprising real-world outcomes (*e.g.* weaker privacy protection than expected for row B above).

Addressing this situation requires modifying the definition of differential privacy to take correlations into account, and add extra noise when correlations allow the adversary to make additional inferences about the sensitive data. The Pufferfish framework (Kifer and Machanavajjhala, 2014) was designed specifically for building such modifications; Pufferfish privacy allows for the specification of a set of data-generating distributions (which may encode correlations in the data) and sets of *secrets* that the privacy definition should protect. Pufferfish privacy ensures that these secrets receive the desired level of privacy protection for every distribution of databases in the specified set (even when correlated data is present).

Instances of Pufferfish privacy are challenging to construct and reason about, because formal descriptions of both the set of secrets and distributions over databases are less intuitive than the broad guarantee provided by differential privacy. *Domain-specific* privacy definitions, grounded in the Pufferfish framework, may offer a promising path

towards addressing this challenge; such definitions can be specifically designed to match the informal expectations of privacy for a particular setting. This approach has been successfully applied to social networks (Liu *et al.*, 2016) and streaming data (Song *et al.*, 2017).

10.2 System Implementation & Integration

The theory of differential privacy is being translated into practical systems for its deployment at an accelerating rate, and this process has uncovered a number of interesting challenges specific to the practical implementation of differentially private mechanisms. We summarize some of these challenges here, in addition to the ones mentioned in Chapters 7, 8, and 9.

Floating-point arithmetic and random number generators. The mathematical definitions of differential privacy mechanisms described in this book assume that generating high-quality random samples from a specific distribution is easy, and that arithmetic is exact—but in actual implementations, neither assumption holds. In fact, the textbook method for generating Laplace noise using standard floating-point arithmetic fails to satisfy differential privacy (Mironov, 2012). Solutions to this problem have been developed and implemented in most popular libraries for differential privacy. In addition, high-quality (cryptographic) random number generators should ideally be used to generate the uniform randomness used in differentially private algorithms; for example, Garfinkel and Leclerc (Garfinkel and Leclerc, 2020) sketch a potential attack against noise generated using the MT19937 pseudo-random number generator (Python's default).

Hyperparameters. Many differentially private algorithms (including the ones described in this book) expose hyperparameters to the analyst, and require these to be set properly in order to achieve good results. However, analysts typically have no experience with these hyperparameters, and no idea how to set them—in fact, for some algorithms, good settings are *unknown*. The idea of clipping, described in Section 6.5 and used in many truncation-based algorithms, is a good example—if the

clipping parameter is set too low, then accuracy suffers because of lost data; if it is set too high, then accuracy suffers because more noise is added than necessary.

As these algorithms are transferred into deployable systems for differential privacy, free hyperparameters become a challenge. Many recent algorithms include methods for setting hyperparameters automatically or adaptively; for example, the recent TSens algorithm (Tao *et al.*, 2020) is truncation-based, but uses the Sparse Vector Technique (Chapter 2) to set the truncation parameter automatically. Eliminating hyperparameters entirely is difficult. But as differential privacy gains widespread use, approaches like this one are likely to become more common, since they make algorithms much easier to use.

Privacy budgeting. Two open challenges in privacy budgeting for differential privacy remain: how to set the privacy budget (discussed in the next section) and how to remain below it. The latter challenge is often a particular concern when deploying differentially private solutions, since when the budget is used up, *the data must be thrown away!* Almost by definition, then, the goal of differentially private algorithm design is to answer more queries with less budget, and stretch the budget out. Even with these techniques, however, it seems almost inevitable that the privacy budget for a dataset will eventually be used up.

As a result, deployments of differential privacy have been primarily focused on discrete releases of data—like the statistics released by the US Census Bureau (Abowd, 2018). Systems designed for differentially private analysis of data often use simplistic methods to account for the budget used over time, and our lack of experience with these tools means we do not yet know how best to set them up to ensure appropriate use of the budget. Progress on the problem of privacy budgeting is therefore likely to require close collaboration between algorithm designers, system builders, and practitioners.

10.3 Social Considerations

This book primarily addresses the technical challenges of differential privacy, and ignores the social impacts and considerations that go with

it. These aspects deserve careful consideration, however, and many of the open challenges in adopting differential privacy lie in understanding the real-world implications of the privacy guarantee and explaining those implications to non-experts. We briefly survey some of these considerations in this section.

Understanding the privacy guarantee. As described in Chapter 1, one way to interpret the guarantee of differential privacy is: *if you participate in a differentially private analysis of data, you will not suffer any additional harm as a result.* This interpretation is both a strength and a challenge associated with differential privacy; it does not require any understanding of the format of the underlying data or correlations between data samples, but it remains challenging to explain the role of the privacy parameter ϵ or the effect of correlations in the data on the outcomes of differentially private analysis in practice.

In industrial deployments, differential privacy is often described as "magical math" that protects privacy "better" than traditional approaches. Progressing beyond this simple description, towards a real understanding of the benefits and challenges of differential privacy among the general public, remains a major challenge. Understanding the nuances of the definition of differential privacy, and of the differences between formal definitions (*e.g.* pure versus approximate differential privacy), adds to this challenge; more complicated definitions (*e.g.* Pufferfish privacy) further complicate the process.

Setting and communicating the privacy budget. Setting ϵ remains a big challenge. Values between 0.1 and 1.0 are often considered reasonable (Dwork, 2011); because for $\epsilon \gg 1$, the probability of harm to an individual increases quickly. It does seem clear that setting $\epsilon \leq 1$ is likely to do a good job protecting privacy in most settings, but it remains *unclear* how much larger ϵ can be before practical harms occur. For example, Apple's differential privacy system has used values for ϵ as high as 16 (Tang *et al.*, 2017), but no practical privacy attack has ever been demonstrated against the system.

The effect of the privacy parameter on real-world privacy outcomes is still an open question, and no clear guidance exists on how to set

the privacy parameter for a given use case. One approach is a kind of "ϵ marketplace" (Dwork, 2011): data curators will compete to offer the lowest ϵ for a given analysis task, and acceptable values of ϵ will naturally fall over time to reach a minimum. Another approach is to allow analysts to specify a desired level of *accuracy*, and automatically set ϵ to the minimum possible value which yields the desired accuracy (*e.g.* Hsu *et al.*, 2014; Lee and Clifton, 2011). Frameworks like APEx (Ge *et al.*, 2019) have been developed to help analysts in this task.

Communicating the implication of a particular setting of ϵ poses an additional challenge. The "meaning" of ϵ is similar to the "meaning" of the security parameter in a cryptosystem—both are difficult to interpret without deep technical knowledge of the underlying theory. In cryptography, the solution is to ask experts to define "reasonable defaults" for the rest of us to use; the analogous defaults (*e.g.* values for ϵ) have yet to be developed in differential privacy.

Supporting analysts. Analysts are often not experts in differential privacy, but the properties of differentially private algorithms can have significant impact on the quality of the results they compute. Future systems for differentially private analysis will need to support analysts in making decisions about the privacy budget and other hyperparameters (as described above), and also about which analyses are reasonable to conduct and how the results should be interpreted (*e.g.* how to deal with noisy results). Systems like PSI (Murtagh *et al.*, 2018) and APEx (Ge *et al.*, 2019) hint at the future of support for analysts. Both provide tools that help the analyst decide what queries to run, how to run them, and how much of the privacy budget to allocate to each one.

Differential privacy and the law. In response to the rapid loss of data privacy over the past decade, governments have begun developing new privacy regulations (*e.g.* the General Data Protection Regulation (GDPR) in the European Union). These regulations recognize the importance of privacy and attempt to specify how it must be protected.

Where does differential privacy fit in the new world of regulated privacy? Unfortunately, it depends on your interpretation of the law.

Regulations like GDPR do not specifically require the use of differential privacy, or any specific privacy definition—and perhaps they should not, in case problems are found in the definition, or a better definition is developed. However, differential privacy is increasingly viewed by organizations as an important tool for complying with regulations like GDPR, and regulators are increasingly aware of the drawbacks of traditional approaches to privacy (Ohm, 2009). In the future, differential privacy may play a larger role both in efforts to comply with existing regulation and in influencing new regulation.

References

Abadi, M., A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. (2016). "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 308–318.

Abowd, J. M. (2018). "The US Census Bureau adopts differential privacy". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2867–2867.

Agarwal, A., M. Herlihy, S. Kamara, and T. Moataz. (2018). "Encrypted Databases for Differential Privacy". In: *IACR Cryptology ePrint Archive*.

Agarwal, S., B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. (2013). "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data". In: *EuroSys*.

Arapinis, M., D. Figueira, and M. Gaboardi. (2016). "Sensitivity of Counting Queries". In: *ICALP*. 120:1–120:13.

Balle, B., G. Barthe, and M. Gaboardi. (2018). "Privacy Amplification by Subsampling: Tight Analyses via Couplings and Divergences". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc. 6277–6287. URL: http://papers.nips.cc/paper/7865-privacy-amplification-by-subsampling-tight-analyses-via-couplings-and-divergences.pdf.

Balle, B., J. Bell, A. Gascon, and K. Nissim. (2020). "Private summation in the multi-message shuffle model". *arXiv preprint arXiv:2002.00817*.

Balle, B., J. Bell, A. Gascón, and K. Nissim. (2019). "The privacy blanket of the shuffle model". In: *Annual International Cryptology Conference*. Springer. 638–667.

Barthe, G., G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P. Strub. (2016). "Differentially Private Bayesian Programming". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM. 68–79. DOI: 10.1145/2976749.2978371.

Bater, J., G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. (2017). "SMCQL: Secure Querying for Federated Databases". 10(6): 673–684. DOI: 10.14778/3055330.3055334.

Bittau, A., Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and B. Seefeld. (2017). "Prochlo: Strong Privacy for Analytics in the Crowd". In: *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. 441–459.

Blocki, J., A. Blum, A. Datta, and O. Sheffet. (2013). "Differentially private data analysis of social networks via restricted sensitivity". In: *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*. Ed. by R. D. Kleinberg. ACM. 87–96. DOI: 10.1145/2422436.2422449.

Bun, M., C. Dwork, G. N. Rothblum, and T. Steinke. (2018). "Composable and versatile privacy via truncated CDP". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM. 74–86.

Bun, M. and T. Steinke. (2016). "Concentrated differential privacy: Simplifications, extensions, and lower bounds". In: *Theory of Cryptography Conference*. Springer. 635–658.

Chan, T. H., E. Shi, and D. Song. (2010). "Private and continual release of statistics". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 405–417.

Chatzikokolakis, K., M. E. Andrés, N. E. Bordenabe, and C. Palamidessi. (2013). "Broadening the Scope of Differential Privacy Using Metrics". In: *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*. Ed. by E. D. Cristofaro and M. K. Wright. Vol. 7981. *Lecture Notes in Computer Science*. Springer. 82–102. DOI: 10.1007/978-3-642-39077-7_5.

Chaum, D. L. (1981). "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". *Commun. ACM*. 24(2): 84–90. DOI: 10.1145/358549.358563.

Chen, S. and S. Zhou. (2013). "Recursive Mechanism: Towards Node Differential Privacy and Unrestricted Joins". In: *ACM SIGMOD*.

Chen, Y., A. Machanavajjhala, M. Hay, and G. Miklau. (2017). "Pegasus: Data-adaptive differentially private stream processing". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1375–1388.

Cummings, R., S. Krehbiel, K. A. Lai, and U. T. Tantipongpipat. (2018). "Differential Privacy for Growing Databases". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. 8878–8887. URL: https://proceedings.neurips.cc/paper/2018/hash/ac27b77292582bc293a51055bfc994ee-Abstract.html.

Dinur, I. and K. Nissim. (2003). "Revealing information while preserving privacy". In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 202–210.

Dwork, C. (2006). "Differential Privacy". In: *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*. Vol. 4052. *Lecture Notes in Computer Science*. Springer Verlag. 1–12. URL: https://www.microsoft.com/en-us/research/publication/differential-privacy/.

Dwork, C. (2011). "A firm foundation for private data analysis". *Commun. ACM*. 54(1): 86–95. DOI: 10.1145/1866739.1866758.

Dwork, C., K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. (2006a). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In: *EUROCRYPT*. Ed. by S. Vaudenay. Vol. 4004. *Lecture Notes in Computer Science*. Springer. 486–503. URL: http://dblp.uni-trier.de/db/conf/eurocrypt/eurocrypt2006.html#DworkKMMN06.

Dwork, C. and J. Lei. (2009). "Differential privacy and robust statistics". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by M. Mitzenmacher. ACM. 371–380. DOI: 10.1145/1536414.1536466.

Dwork, C., F. McSherry, K. Nissim, and A. Smith. (2006b). "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Ed. by S. Halevi and T. Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg. 265–284.

Dwork, C., M. Naor, T. Pitassi, and G. N. Rothblum. (2010). "Differential Privacy Under Continual Observation". In: *Proceedings of the Forty-second ACM Symposium on Theory of Computing. STOC '10*.

Dwork, C., M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan. (2009). "On the Complexity of Differentially Private Data Release: Efficient Algorithms and Hardness Results". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. STOC '09*. Bethesda, MD, USA: Association for Computing Machinery. 381–390. DOI: 10.1145/1536414.1536467.

Dwork, C., A. Roth, *et al.* (2014). "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*. 9(3-4): 211–407.

Ebadi, H., D. Sands, and G. Schneider. (2015). "Differential Privacy: Now it's Getting Personal". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. Ed. by S. K. Rajamani and D. Walker. ACM. 69–81. DOI: 10.1145/2676726.2677005.

Erlingsson, Ú., V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. (2019). "Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity". *Annual ACM-SIAM Symposium on Discrete Algorithms*: 2468–2479.

Erlingsson, Ú., V. Pihur, and A. Korolova. (2014). "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. 1054–1067. DOI: 10.1145/2660267.2660348.

Gaboardi, M., E. J. G. Arias, J. Hsu, A. Roth, and Z. S. Wu. (2014). "Dual query: Practical private query release for high dimensional data". In: *International Conference on Machine Learning*. 1170–1178.

Garfinkel, S. L. and P. Leclerc. (2020). "Randomness Concerns when Deploying Differential Privacy". In: *WPES'20: Proceedings of the 19th Workshop on Privacy in the Electronic Society, Virtual Event, USA, November 9, 2020*. Ed. by J. Ligatti, X. Ou, W. Lueks, and P. Syverson. ACM. 73–86. DOI: 10.1145/3411497.3420211.

Ge, C., X. He, I. F. Ilyas, and A. Machanavajjhala. (2019). "APEx: Accuracy-Aware Differentially Private Data Exploration". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska. ACM. 177–194.

Ghazi, B., P. Manurangsi, R. Pagh, and A. Velingker. (2020). "Private aggregation from fewer anonymous messages". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 798–827.

Haeberlen, A., B. C. Pierce, and A. Narayan. (2011). "Differential Privacy Under Fire". In: *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association. URL: http://static.usenix.org/events/sec11/tech/full%5C_papers/Haeberlen.pdf.

Hardt, M., K. Ligett, and F. McSherry. (2012). "A simple and practical algorithm for differentially private data release". In: *Advances in Neural Information Processing Systems*. 2339–2347.

Hay, M., A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. (2016a). "Principled evaluation of differentially private algorithms using dpbench". In: *Proceedings of the 2016 International Conference on Management of Data*. 139–154.

Hay, M., A. Machanavajjhala, G. Miklau, Y. Chen, D. Zhang, and G. Bissias. (2016b). "Exploring Privacy-Accuracy Tradeoffs using DPComp". In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by F. Özcan, G. Koutrika, and S. Madden. ACM. 2101–2104. DOI: 10.1145/2882903.2899387.

He, X., A. Machanavajjhala, and B. Ding. (2014a). "Blowfish privacy: tuning privacy-utility trade-offs using policies". In: *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. Ed. by C. E. Dyreson, F. Li, and M. T. Özsu. ACM. 1447–1458. DOI: 10.1145/2588555.2588581.

He, X., A. Machanavajjhala, and B. Ding. (2014b). "Blowfish privacy: tuning privacy-utility trade-offs using policies". In: *ACM SIGMOD*. 1447–1458.

Hsu, J., M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth. (2014). "Differential Privacy: An Economic Method for Choosing Epsilon". In: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society. 398–410. DOI: 10.1109/CSF.2014.35.

"IBM Differential Privacy Library". https://github.com/IBM/differential-privacy-library.

Ilvento, C. (2020). "Implementing the Exponential Mechanism with Base-2 Differential Privacy". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by J. Ligatti, X. Ou, J. Katz, and G. Vigna.

Johnson, N. M., J. P. Near, J. M. Hellerstein, and D. Song. (2020a). "Chorus: a Programming Framework for Building Scalable Differential Privacy Mechanisms". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE. 535–551. DOI: 10.1109/EuroSP48549.2020.00041.

Johnson, N. M., J. P. Near, J. M. Hellerstein, and D. Song. (2020b). "Chorus: a Programming Framework for Building Scalable Differential Privacy Mechanisms". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE. 535–551. DOI: 10.1109/EuroSP48549.2020.00041.

Johnson, N. M., J. P. Near, and D. Song. (2018). "Towards Practical Differential Privacy for SQL Queries". *Proc. VLDB Endow.* 11(5): 526–539. DOI: 10.1145/3187009.3177733.

Karwa, V., S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. (2011). "Private Analysis of Graph Structure". In: *PVLDB*.

Kasiviswanathan, S. P., H. K. Lee, K. Nissim, S. Raskhodnikova, and A. D. Smith. (2011). "What Can We Learn Privately?" *SIAM J. Comput.* 40(3): 793–826. DOI: 10.1137/090756090.

Kasiviswanathan, S. P., K. Nissim, S. Raskhodnikova, and A. Smith. (2013a). "Analyzing Graphs with Node Differential Privacy". In: *TCC*.

Kasiviswanathan, S. P., K. Nissim, S. Raskhodnikova, and A. D. Smith. (2013b). "Analyzing Graphs with Node Differential Privacy". In: *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Ed. by A. Sahai. Vol. 7785. *Lecture Notes in Computer Science*. Springer. 457–476. DOI: 10.1007/978-3-642-36594-2_26.

Kifer, D. and A. Machanavajjhala. (2014). "Pufferfish: A framework for mathematical privacy definitions". *ACM Trans. Database Syst.* 39(1): 3:1–3:36. DOI: 10.1145/2514689.

Korolova, A., K. Kenthapadi, N. Mishra, and A. Ntoulas. (2009). "Releasing search queries and clicks privately". In: *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*. Ed. by J. Quemada, G. León, Y. S. Maarek, and W. Nejdl. ACM. 171–180. DOI: 10.1145/1526709.1526733.

Kotsogiannis, I., Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. (2019). "PrivateSQL: A Differentially Private SQL Query Engine". *Proc. VLDB Endow.* 12(11): 1371–1384. DOI: 10.14778/3342263.3342274.

Lee, J. and C. Clifton. (2011). "How Much Is Enough? Choosing ϵ for Differential Privacy". In: *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*. Ed. by X. Lai, J. Zhou, and H. Li. Vol. 7001. *Lecture Notes in Computer Science*. Springer. 325–340. DOI: 10.1007/978-3-642-24861-0_22.

Lee, J. and C. W. Clifton. (2014). "Top-k Frequent Itemsets via Differentially Private FP-Trees". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '14*. New York, New York, USA: Association for Computing Machinery. 931–940. DOI: 10.1145/2623330.2623723.

Li, C., G. Miklau, M. Hay, A. McGregor, and V. Rastogi. (2015). "The matrix mechanism: optimizing linear counting queries under differential privacy". *VLDB J.* 24(6): 757–781. DOI: 10.1007/s00778-015-0398-x.

Liu, C., S. Chakraborty, and P. Mittal. (2016). "Dependence Makes You Vulnerable: Differential Privacy Under Dependent Tuples". In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/dependence-makes-you-vulnerable-differential-privacy-under-dependent-tuples.pdf.

Lu, W., G. Miklau, and V. Gupta. (2014). "Generating Private Synthetic Databases for Untrusted System Evaluation". In: *ICDE*.

Lyu, M., D. Su, and N. Li. (2017). "Understanding the Sparse Vector Technique for Differential Privacy". *Proc. VLDB Endow.*

Machanavajjhala, A., D. Kifer, J. Gehrke, and M. Venkitasubramaniam. (2007). "l-diversity: Privacy beyond k-anonymity". *ACM Transactions on Knowledge Discovery from Data (TKDD)*. 1(1): 3–es.

McGregor, A., I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan. (2010). "The Limits of Two-Party Differential Privacy". In: *Annual Symposium on Foundations of Computer Science*. IEEE.

McKenna, R., R. K. Maity, A. Mazumdar, and G. Miklau. (2020). "A workload-adaptive mechanism for linear queries under local differential privacy". *Proc. VLDB Endow.* 13(11): 1905–1918. URL: http://www.vldb.org/pvldb/vol13/p1905-mckenna.pdf.

McKenna, R., G. Miklau, M. Hay, and A. Machanavajjhala. (2018). "Optimizing error of high-dimensional statistical queries under differential privacy". *PVLDB*. 11(10): 1206–1219. DOI: 10.14778/3231751.3231769.

McKenna, R., D. Sheldon, and G. Miklau. (2019). "Graphical-model based estimation and inference for differential privacy". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. *Proceedings of Machine Learning Research*. PMLR. 4435–4444. URL: http://proceedings.mlr.press/v97/mckenna19a.html.

McSherry, F. D. (2009). "Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. SIGMOD '09*. Providence, Rhode Island, USA. 19–30.

McSherry, F. and K. Talwar. (2007). "Mechanism Design via Differential Privacy". In: *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE. URL: https://www.microsoft.com/en-us/research/publication/mechanism-design-via-differential-privacy/.

Mironov, I. (2012). "On significance of the least significant bits for differential privacy". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 650–661.

Mironov, I. (2017). "Renyi differential privacy". In: *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*. IEEE. 263–275.

Mironov, I., O. Pandey, O. Reingold, and S. Vadhan. (2009). "Computational Differential Privacy". In: *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '09*. Santa Barbara, CA: Springer-Verlag. 126–142. DOI: 10.1007/978-3-642-03356-8_8.

Mohan, P., A. Thakurta, E. Shi, D. Song, and D. E. Culler. (2012). "GUPT: privacy preserving data analysis made easy". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. 349–360. DOI: 10.1145/2213836.2213876.

Murtagh, J., K. Taylor, G. Kellaris, and S. Vadhan. (2018). "Usable Differential Privacy: A Case Study with PSI". arXiv: 1809.04103 [cs.HC].

Narayan, A. and A. Haeberlen. (2012). "DJoin: Differentially Private Join Queries over Distributed Databases". In:

Nissim, K., S. Raskhodnikova, and A. D. Smith. (2007). "Smooth sensitivity and sampling in private data analysis". In: *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*. Ed. by D. S. Johnson and U. Feige. ACM. 75–84. DOI: 10.1145/1250790.1250803.

Ohm, P. (2009). "Broken promises of privacy: Responding to the surprising failure of anonymization". *UCLA l. Rev.* 57: 1701.

"OpenDP". https://privacytools.seas.harvard.edu/opendp.

Raskhodnikova, S. and A. D. Smith. (2015). "Efficient Lipschitz Extensions for High-Dimensional Graph Statistics and Node Private Degree Distributions". *CoRR*. abs/1504.07912. arXiv: 1504.07912. URL: http://arxiv.org/abs/1504.07912.

Roth, E., D. Noble, B. H. Falk, and A. Haeberlen. (2019). "Honeycrisp: large-scale differentially private aggregation without a trusted core". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*. Ed. by T. Brecht and C. Williamson. ACM. 196–210. DOI: 10.1145/3341301.3359660.

Roy, I., S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. (2010). "Airavat: Security and Privacy for MapReduce". In: *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*. USENIX Association. 297–312. URL: http://www.usenix.org/events/nsdi10/tech/full%5C_papers/roy.pdf.

"Smart Noise". https://github.com/opendp/smartnoise-samples.

Smith, A. D. (2011). "Privacy-preserving statistical estimation with optimal convergence rates". In: *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM. 813–822. DOI: 10.1145/1993636.1993743.

Song, S., Y. Wang, and K. Chaudhuri. (2017). "Pufferfish Privacy Mechanisms for Correlated Data". In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu. ACM. 1291–1306. DOI: 10.1145/3035918.3064025.

Sweeney, L. (2000). "Simple demographics often identify people uniquely". *Health (San Francisco)*. 671(2000): 1–34.

Sweeney, L. (2002). "k-anonymity: A model for protecting privacy". *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*. 10(05): 557–570.

Tang, J., A. Korolova, X. Bai, X. Wang, and X. Wang. (2017). "Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12". *CoRR*. abs/1709.02753. arXiv: 1709.02753. URL: http://arxiv.org/abs/1709.02753.

Tao, Y., X. He, A. Machanavajjhala, and S. Roy. (2020). "Computing Local Sensitivities of Counting Queries with Joins". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo. ACM. 479–494. DOI: 10.1145/3318464.3389762.

Vesga, E. L., A. Russo, and M. Gaboardi. (2020). "A Programming Framework for Differential Privacy with Accuracy Concentration Bounds". In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE. 411–428. DOI: 10.1109/SP40000.2020.00086.

Wang, T., J. Blocki, N. Li, and S. Jha. (2017). "Locally Differentially Private Protocols for Frequency Estimation". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by E. Kirda and T. Ristenpart. USENIX Association. 729–745. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tianhao.

Wang, T., B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha. (2019). "Answering Multi-Dimensional Analytical Queries under Local Differential Privacy". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 159–176. DOI: 10.1145/3299869.3319891.

Warner, S. L. (1965). "Randomized response: A survey technique for eliminating evasive answer bias". *Journal of the American Statistical Association*. 60(309): 63–69.

Wilson, R. J., C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. (2020). "Differentially Private SQL with Bounded User Contribution". *Proceedings on Privacy Enhancing Technologies*. 2020(2): 230–250.

Yang, J., T. Wang, N. Li, X. Cheng, and S. Su. (2020). "Answering Multi-Dimensional Range Queries under Local Differential Privacy". *CoRR*. abs/2009.06538. arXiv: 2009.06538. URL: https://arxiv.org/abs/2009.06538.

Zhang, D., R. McKenna, I. Kotsogiannis, G. Bissias, M. Hay, A. Machanavajjhala, and G. Miklau. (2020). "ϵKTELO: A Framework for Defining Differentially Private Computations". *ACM Trans. Database Syst.*

Zhang, D. and D. Kifer. (2017). "LightDP: towards automating differential privacy proofs". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. Ed. by G. Castagna and A. D. Gordon. ACM. 888–901. URL: http://dl.acm.org/citation.cfm?id=3009884.

Zhang, J., G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. (2014). "PrivBayes: private data release via bayesian networks". In: *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. 1423–1434. DOI: 10.1145/2588555.2588573.

Zhang, Z., T. Wang, N. Li, S. He, and J. Chen. (2018). "CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by D. Lie, M. Mannan, M. Backes, and X. Wang. ACM. 212–229. DOI: 10.1145/3243734.3243742.