Summer 2014

Privacy-preserving Sanitization in Data Sharing

Wentian Lu

PRIVACY-PRESERVING SANITIZATION IN DATA SHARING

A Dissertation Presented

by

WENTIAN LU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

Computer Science

PRIVACY-PRESERVING SANITIZATION IN DATA SHARING

A Dissertation Presented

by

WENTIAN LU

Approved as to style and content by:

Gerome Miklau, Chair

Neil Immerman, Member

David Jensen, Member

Krista Gile, Member

Lori Clarke, Department Chair
Computer Science

ACKNOWLEDGMENTS

This work would not have been possible without the guidance and support of Gerome Miklau. With his patience and tireless effort, I have developed myself as a passionate researcher. I would thank him for sharing his experience, teaching me about research skills and helping me shape this work. I am also grateful for his effort on editing and improving my writings.

I'd like to thank Neil Immerman for his input and insight of our collaborated work. His passion about scientific inquiry always inspires me. I would also like to thank my other committee members, David Jensen and Krista Gile for their technical suggestions and valuable comments on improving the presentation of this dissertation. I am also grateful to other professors in the Database group, Yanlei Diao and Alexandra Meliou. I learned a lot from their professions and experience.

During my time in Amherst, I have benefitted greatly from the support of all of my fellow graduate students who helped me through the journey, especially everyone in the Database group. I am thankful for your encouragement and friendship.

I'd like to thank Leeanne Leclerc and Rachel Lavery for their help and effort to make our graduate students' life much easier. I also thank the faculty and staff for providing a supportive learning and working environment in the department.

Finally, I am greatly indebted to my family. I would like to thank my wife, Shan, for her continuous love and support. Finishing this journey is never possible without her encouragement. My parents have always supported me for pursuing my interests, even to the other side of the planet, for which I will always be grateful.

ABSTRACT

PRIVACY-PRESERVING SANITIZATION IN DATA SHARING

SEPTEMBER 2014

WENTIAN LU

B.S., NANJING UNIVERSITY

M.S., NANJING UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Gerome Miklau

In the era of big data, the prospect of analyzing, monitoring and investigating all sources of data starts to stand out in every aspect of our life. The benefit of such practices becomes concrete only when analysts or investigators have the information shared from data owners. However, privacy is one of the main barriers that disrupt the sharing behavior, due to the fear of disclosing sensitive information. This dissertation describes data sanitization methods that disguise the sensitive information before sharing a dataset and our criteria are always protecting privacy while preserving utility as much as possible.

In particular, we provide solutions for tasks that require different types of shared data. In the case of sharing partial content of a dataset, we consider the problem of releasing a database under retention restrictions such that the auditing job can

still be carried out. While obeying a retention policy often results in the wholesale destruction of the audit log in existing solutions, our framework allows to expire data at a fine granularity and supports audit queries on a database with incompleteness. Secondly, in the case of sharing the entire dataset, we solve the problem of untrusted system evaluation using released database synthesis under differential privacy. Our synthetic database accurately preserves the core performance measures of a given query workload, and satisfies differential privacy with crucial extensions to multi-relation databases. Lastly, in the case of sharing derived information from the data source, we focus on distributing results of network modeling under differential privacy. Our mechanism can safely output estimated parameters of the exponential random graph model, by employing a decomposition of the estimation problem into two steps: getting private sufficient statistics first and then estimating the model parameters. We show that our privacy mechanism provides provably less error than common baselines and our redesigned estimation algorithm offers better accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

With the emergence of mobile devices, internet and social networks, vast numbers of documents and datasets are maintained by companies from a range of industries such as healthcare, insurance and information technology. These datasets are highly valuable as they contain information about persons. Not surprisingly, there is increasing need from governments, researchers and even the inside of enterprises that requests access to the data for monitoring, understanding and analyzing purpose. However, privacy concerns could be one of the main barriers that disrupt such behaviors. So the question of how to share these data but still preserve privacy is an important and realistic problem in the real world.

Researchers in the privacy community propose mechanisms to share information from various data sources while preventing the disclosure of the sensitive parts. The sensitive information could either be explicitly specified by a declarative language, e.g., access control [85], or indicated by the semantics of privacy definition, e.g., differential privacy [23]. Before sharing a dataset that contains sensitive information, the process of *data sanitization* disguises sensitive information and replace it with non-identifiable values.

1.1 Problem setting

A typical scenario of data sharing involves two parties: data owner and data consumer, where data owner shares the data and data consumer consumes data, as

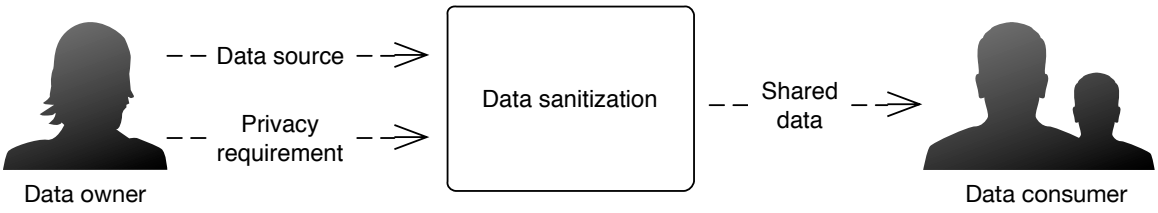


Figure 1.1: A typical scenario of data sharing

shown in Figure 1.1. For data owners, they need two clearly defined objects, data source and privacy requirement.

Data source is the actual target of sanitization, which is the first thing that should be decided. A precise definition of data source helps to further investigate potential threats and possible methods for manipulating the data later. In the real world, there is a wide range of data sources in a sanitization process, e.g., text documents, relational databases and social networks.

The purpose of sanitization is to protect sensitive information from being disclosed. The *privacy requirement* defines the sensitive contents of the current data. Without suitable insight of sensitive information, people cannot illustrate or prove the correctness of a sanitized output. A clear definition of sensitive contents will also help to understand the underlying sanitizing algorithms when end users look for a particular technique on their data sets. In the literature, there are several commonly accepted definition for sensitive information. For example, for tabular data, access control [85] pins down sensitive information by predefined tables or views; differential privacy [23] treats each individual's existence as a sensitive object.

For a data consumer, we assume that the only way to access the data source is through a sanitization interface, as shown in Figure 1.1. As there is no restrictions on how shared data can be used, our major focus is to guarantee the released data satisfy privacy requirements, event though a data consumer could be potentially malicious.

The most challenging part is designing a proper method for sanitizing data. Good sanitization should always care about utility, otherwise we can just release an empty dataset which prevents any information leakage. If we consider the potential analysis applied on the shared data, the output of sanitization should ease the future computation and do as much as possible to reduce the damage to the utility. So the key of sanitizing algorithms is to achieve as much utility as we can, while still guaranteeing the correctness of privacy protection.

The output of sanitization process, *shared data*, could be either homogenous or heterogeneous to the data source. We say the output is homogenous when the shared data is syntactically close to the data source, meaning that both are represented by one data model so that queries can be answered with both data in a similar way. Consider the case when the data source is a relational database. As long as the shared data are tables under the relational data model, they are homogenous as the well-established query evaluation techniques can be applied on the shared data. In this thesis, we study two problems that aim for releasing homogenous data, sharing partial contents of the data source and sharing the entire data source. On the other hand, the output of sanitizing process could be fundamentally different from the data source, where the typical operations on the shared data are distinct from operations on the data source. In such cases, we say the shared data are derived or heterogeneous. An example is to release the estimated power law exponent from the degree sequence of a network. In this thesis, we solve one problem with the goal of releasing derived data from a network.

1.2 Overview of contributions

1.2.1 Sharing partial data

Chapter 3 addresses the data sanitization problem when a data owner is sharing partial contents of the data source. In particular, the privacy requirement demands

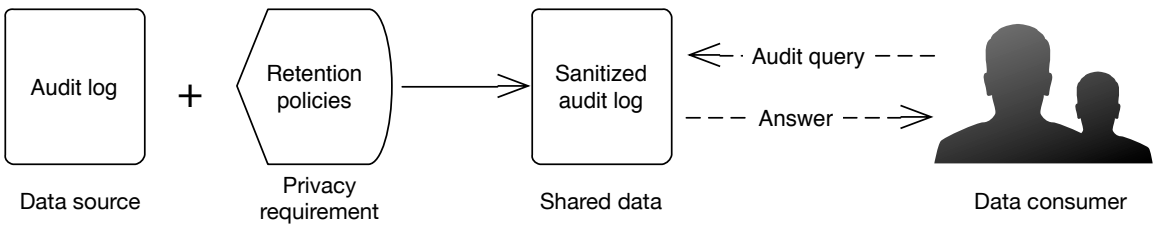


Figure 1.2: Auditing a database with retention policies

protection of a subset of information in the data source, while the rest is safe for releasing.

We consider the scenario of auditing the changes to a database, where auditors (data consumers) submit audit queries and inquire about what happened to the database, when it happened and who did it. But an accurate audit log is an historical record of the past that can also pose a serious threat to privacy. In many domains, retention policies are introduced to govern how long data can be preserved by an institution. Data owners often adopt their own policies for the purpose of limiting retention and removing sensitive data after a period of time to avoid unintended release. Our goal is to audit a database system in the presence of retention restrictions. As explained in Figure 1.2, in the sanitization-based approach, an audit log is sanitized with the enforcement of retention policies while data consumers are restricted to submit audit queries to the shared sanitized data.

Though the goal of sanitizing is to limit the conflicts between private information (limited retention) and sharing information (insensitive database changes), unfortunately, existing mechanisms for auditing and managing historical records have few capabilities for managing the balance between these two objectives. Obeying a retention policy often means the wholesale destruction of the audit log.

In Chapter 3, we will illustrate the importance of our framework that properly models the data source, which allows expressive privacy requirement specification and efficient data sanitization. An effective data model not only defines the data semantics

eid	name	dept	sal	from	to
101	Bob	Sales	10	0	200
101	Bob	Mgmt	10	200	300
101	Bob	Mgmt	15	300	now
201	Chris	HR	8	0	300

Table 1.1: Data source: relational data model with temporal information

eid	name	dept	sal	from	to
101	Bob	Sales	**sx**	0	200
101	Bob	Mgmt	**sx**	200	300
101	Bob	Mgmt	15	300	now
201	Chris	HR	8	0	300

Table 1.2: Sanitized data under retention policies

after sanitization but also determines the scope of manipulation on the data. In our work, we propose a relational data model that can deal with historical information and incompleteness, which supports temporal-based operations and uncertainty in the query results. To be precise about private information and retain as much utility as possible, it is important that the privacy requirement is specified in a flexible way. We propose a rule-based expressive language for data owners to define retention policies at the granularity of attributes. Under retention policies, the audit history is partially incomplete. Thus, audit queries on the protected history can include imprecise results and our challenge is to solve the problem of representing and computing such imprecise answers over incomplete databases. Chapter 3 illustrates that the combination of data model and policy language is the strong basis for auditing purpose. Despite of removal of information, our query answering system in many cases enables an auditor to monitor the record of actions taken on the database.

Example 1.1. Table 1.1 is a transaction-time table that represents the complete data history of the table, where the **from** and **to** columns is the active period of each tuple in the database.

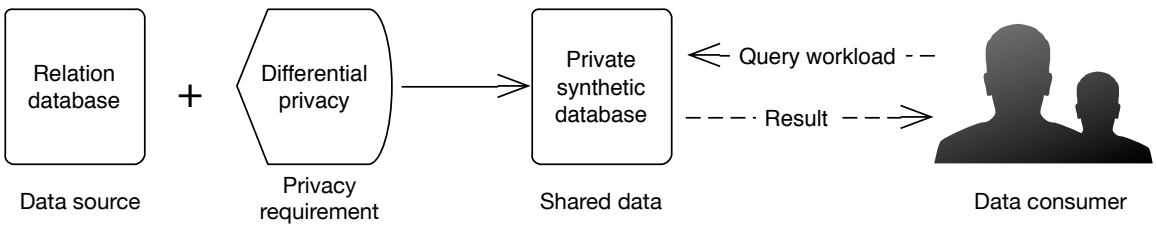


Figure 1.3: Sharing private synthetic database

Now consider two different retention polices. One is to hide attribute values while revealing the existence of the tuple, e.g., hiding Bob's salary between time 0 and 300. The other is to completely remove the tuple, e.g., removing the record of all employees in the HR department between time 0 and 300. The resulting sanitized table (Table 1.2) will have data deleted, particularly, removed tuples (in gray) and substituted variables (in bold). In Chapter 3, we discuss in detail the correctness of this sanitization process and the problem of how to answer audit queries with sanitized tables.

This work appeared in the proceedings of the International Conference on Very Large Data Bases, 2008 [67], the International Conference on Data Engineering, 2009 [68], and in the International Journal on Very Large Data Bases, 2013 [69].

1.2.2 Sharing the whole dataset

Chapter 4 addresses the data sanitization problem when data owners want to share the whole dataset. In particular, we offer solutions of generating private synthetic datasets.

We consider the scenario when database evaluation tasks should be outsourced to untrusted data consumers, where a typical evaluation task is to assess the performance of a given SQL query workload. There is a strong need in industry for synthetic databases, as resorting to common benchmark databases (e.g. a TPC benchmark)

doesn't help. Because benchmarks target the common case, they often cannot reflect particular properties that may significantly impact performance for a given enterprise.

The data source in this problem is a standard relational database and the privacy requirement is to protect every individual in the data source, i.e., the shared synthetic database should not disclose information of any individual who involves in the data. To meet the privacy goal, we adopt another privacy standard, *differential privacy*, which guarantees that data consumers cannot distinguish the existence of any person from sanitized data. In fact, differential privacy allows us to enforce more rigorous sanitization and provide a provably private solution. In Figure 1.3, data owners share the database synthesis under differential privacy and data consumers conduct performance evaluation using given query workloads.

Chapter 4 describes a novel and fundamentally different approach compared to Chapter 3, which is called *model-based database synthesis*. Our framework first selects a set of queries that serve as the model of database. From the model, we can compute the statistics of database by answering those selected queries. We then sanitize the statistics. Finally, we sample databases using sanitized statistics and release them. The whole process is proved to satisfy differential privacy, and our experiments show that the shared synthetic database preserves core performance properties of given query workloads.

Our work is a novel combination of research into private data release and synthetic database generation. Generating private synthetic data is a common goal of privacy research, but existing techniques do not support complex relational schemas and have not targeted our specific utility goal: accurate system testing and evaluation. Likewise, generating synthetic relational data is a common goal of relational database research. Privacy concerns are often mentioned as one motivation for the use of synthetic databases, however the vast majority of database generation approaches [2, 7, 13, 48, 65] do not offer any formal privacy guarantees. Instead, they

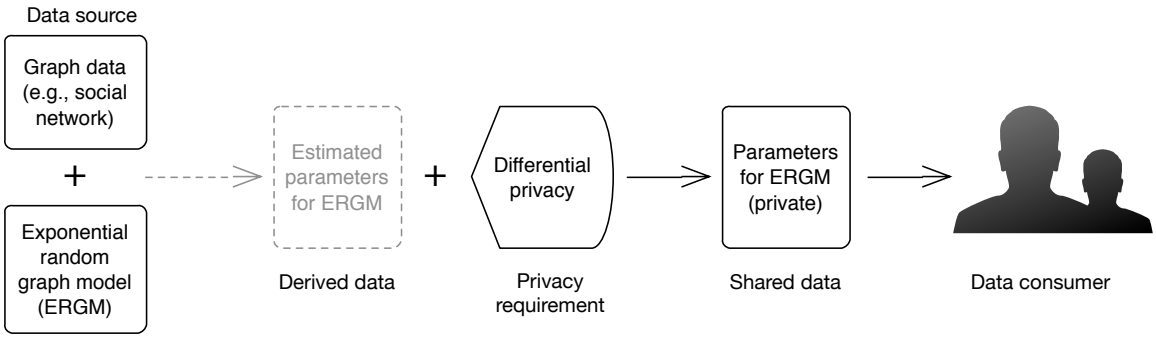


Figure 1.4: Sharing exponential random graph estimation under differential privacy

often rely merely on the fact that data is generated from aggregate statistics about the database. Unfortunately, this does not imply that the synthetic data is safe to release. For example, Arasu et al [2] acknowledge the privacy issues of releasing cardinality information during data generation. One exception is the work of Wu et al. [113], in which cell suppression and perturbation are used to offer some protection against disclosures, but this method cannot satisfy differential privacy and is susceptible to the previously-documented attacks on anonymization schemes.

To achieve differential privacy, we extend its definition to multiple tables by redefining the core concepts. This is a crucial extension for our framework and is useful beyond the present work.

This work appeared in the proceedings of the International Conference on Data Engineering, 2014 [70].

1.2.3 Sharing derived data

Chapter 5 addresses a sanitization problem when the shared information is derived and heterogeneous from the data source.

We consider the scenario of social network analysis under differential privacy, where data consumers are demanding the results of computation on graph-structured data. While differentially private algorithms for computing basic graph properties have been proposed, graph modeling tasks that are common to the data mining

community can not yet be carried out privately. In Chapter 5, we study for the first time the differentially private use of the classic exponential random graph model (ERGM [71]), the most promising and central of social network analysis [19, 28, 25] in recent years. Technically, ERGM describes a parametric statistical model over graphs, and one of most important tasks in modeling practices is to find the best parameters for the observed data source.

A common paradigm that implements differential privacy is output perturbation, i.e., adding noise to the output of computation, while the amount of noise should be carefully adjusted with regard to the properties of the computation. As the shared data in this situation are the parameters of ERGM, a straightforward idea for sanitizing under differential privacy is to execute output perturbation, as illustrated in Figure 1.4. One precomputes parameters (box with dashed boundaries) using standard ERGM algorithms and then add proper noise before sharing them. However, due to internal complexity of ERGM itself, such methods are generally not feasible. To be specific, we are facing the case that either the amount of noise required for differential privacy is extremely hard to compute, or direct noise is so large that perturbed result doesn't make any sense any more.

Our solution in Chapter 5 takes a different approach, employing a decomposition of estimation process into two steps: getting private sufficient statistics first and then estimating the model parameters. The estimation process, a Bayesian based method, is carefully designed, such that the particular noise added into sufficient statistics will be leveraged in the process but still preserves privacy. Compared to Figure 1.4, we save the step for estimating non-private parameters for ERGM (box with dashed boundaries). We consider recent specifications of ERGMs and show that our privacy mechanism offers provably less error than comparable methods. We also illustrate better accuracy by using our redesigned estimation algorithm in the experiments.

1.3 Thesis organization

Chapter 2 reviews the background knowledge related to privacy-preserving data sanitization, including access control, managing incompleteness with historical data and differential privacy. Each of Chapter 3, Chapter 4 and Chapter 5 focuses on the work related to a particular aspect of contributions discussed in previous sections. These chapters also include detailed discussion of problem settings, contributions, solutions and experiments. Chapter 6 concludes with the review of contributions and future work.

CHAPTER 2

BACKGROUND

This chapter provides discussions of background information for this thesis work. We start with access control techniques in database systems, as the problem in Chapter 3 has a similar privacy setting. In both cases, sensitive information is enforced by policies and query answers are generated with compositions of those enforcements. We next review works that focus on managing incompleteness in a temporal database, for the purpose of defining data source in Chapter 3. Last, we discuss a strong and rigorous privacy definition, differential privacy, which is the privacy requirement for Chapter 4 and 5.

2.1 Access control

Access control [85] is the classical technique in database systems to limit the data access of users. The control is implemented by granting users certain access rights (such as read, modify and delete) on predefined objects. In SQL, this is implemented using GRANT command. An example that grants read privilege to user `Ann` on object `Salary` is as follows:

```
GRANT select ON Salary TO Ann
```

The object `Salary` could either be a base table or a view. In database systems, a view is essentially a virtual table defined by a query. For example, the view Salary contains grades from Sales department, generated from the following query:

```
CREATE VIEW Salary (name, salary) AS
        SELECT name, salary
        FROM Employee
        WHERE dept = 'Sales'
```

Queries submitted from clients will have to pass the permission check. If there is not sufficient access privilege, the query will be denied. In database systems, control policies are typically enforced by administrators, who is a trusted party to data owners.

Fine-grained access control As the traditional access control are executed at table-level, there is growing needs for fine-grained control at row-level or even cell-level. One of the major motivations is privacy protection. Lower level controls have the ability of protecting individual information across tuples and cells in the database. For example, the teacher will allow each student to access his own grade in the Grades table. So consider student `Ann` attempts a read as

```
SELECT salary
FROM Employee
```

The database system actually executes:

```
SELECT salary
FROM Employee
WHERE name = 'Ann'
```

In general, a fine-grained access control system is implemented in two steps. Firstly, users specify rules using a policy-defining language. Secondly, the database enforces those rules when answering queries. Several commercial databases provide solutions for fine-grained access control, such as Virtual Private Database (VPD) in Oracle [81] and label-based framework in DB2 [87].

Recently, Wang et. al. [108] proposes a set of theoretical terms that are formalized to guarantee the correctness of the enforcement of fine-grain policies. They are

secrecy, soundness and *maximality*. A mechanism is secure when the query answers do not reveal any information that should be protected by policies. Soundness means the query answers under access control are consistent with answers when no access control is presented. Maximality requires that query evaluation returns as much information as possible. These terms also serve as the basis for our sanitization process in Chapter 3.

2.2 Managing incompleteness with data history

To support retention policies, a database should not only contain current data but also historical information. Such history is usually recognized as temporal data, and database that facilitates managing and operating time-oriented information is called temporal or transaction-time database. In fact, transaction-time databases have been studied extensively by the research community including work on query languages and logical foundations [17, 30, 103], implementation techniques [53, 66, 96], techniques for accommodating time in standard databases [92, 101], as well as implemented extensions to existing systems [102]. Jensen studied querying backlog relations to monitor changes to a database [52].

On the other hand, when enforcing retention policies, a common method is removing expired tuples or attributes, which could potentially generate a database with incomplete history. With the absence of time information, incomplete information also has a long history in databases [9, 37, 51].

When managing retention policies, a data model that handles both temporal data and incompleteness is necessary. Though both areas have been studied extensively, incomplete temporal databases have attracted less attention. The data models of Gadia [31] and Koubarakis [58] establish the foundations of this area.

Gadia's model [31] The model of temporal incompleteness presented by Gadia et al. allows for uncertainty about values, as we do, but also represents certain values

whose active period is uncertain. By combining the different kinds of incomplete information, the model can represent the known and unknown values of an object where its existence could be clear or not.

In Gadia's model, each cell in a table is defined as a set of temporal assignments, i.e., a constant associated with different active time period. For example, the cell of Bob's salary is $\{10[0, 100], 12[101, 200]\}$, which means Bob's salary was 10 from time 0 to 100 and then 12 from 101 to 200. To represent uncertainty, each cell (temporal assignments) is assigned with two temporal periods, l and u. l represents the time period that we are sure it exists, while we are also sure the value beyond u definitely does not exists. Therefore, Bob's salary is noted as a triple (x, l, u), where $x = \{10[0, 100], 12[101, 200]\}$, $l = [0, 150]$, and $u = [0, 300]$. The semantics is as follows:

- $[0, 150]$: we know for sure Bob's salary is 10 before time 100 and 12 after that.
- $[151, 200]$: we are not sure whether Bob's salary exists, but if it exists it should be 12.
- $[201, 300]$: we are not sure existence of Bob's salary and its value.
- $[301, now]$: we are sure Bob's salary does not exist.

Koubarakis' model [58] Koubarakis proposed a constraint-based incomplete temporal data model, which integrates global and local inequality constraints on the occurrence time of an event. With constraints only on temporal information, it supports indefinite instants. An indefinite instant is a very general kind of instant that includes indeterminate instants, instants with disjoint sets of possible chronons, and instants with incompletely specified upper and lower supports.

Bus	Time	Condition
`bus1`	x	$10 \leq x_l, x_r \leq 15$

For example, in the above table, the scheduled time of `bus1` is represented as variable $x = [x_l, x_r]$. The starting time of x, x_l is unknown, but limited to a value at least 10. Similarly, the ending time x_r, uncertain either, but must be at most 15.

In fact, Koubarakis' model can model more complicated scenarios, e.g., we can say this event a happens between time 0 and occurrence of event b, but we do not know its exact time. Such information related to event ordering is not allowed in Gadia's model. However, unlike Gadia's model, without variables in traditional attributes, we cannot represent the information that allows attribute values to be unknown in the (possible and certain) active period of that tuple.

2.3 Differential privacy

When querying a database, differential privacy protects individuals by restricting the impact on the output of any individual opts in or out the database, such that an intruder cannot tell whether any particular individual is in the dataset. Differential privacy [23] is traditionally defined over a tabular based database x consisting of records, each of which describes an individual. Two databases that differ by one record are called *neighbors*. Formally, we say an algorithm is differentially private if distributions of outputs on neighboring databases are unchanged.

Definition 2.1 (Differential Privacy [23])**.** Let x and x' be neighboring databases and $\mathcal{K}$ be any algorithm. For any subset of outputs $O \subseteq \mathit{Range}(\mathcal{K})$, the following holds:

$$\Pr[\mathcal{K}(x) \in O] \leq \exp(\epsilon) \times \Pr[\mathcal{K}(x') \in O] + \delta$$

If $\delta = 0$, $\mathcal{K}$ is standard ϵ-differentially private. Otherwise, $\mathcal{K}$ is relaxed (ϵ, δ)-differentially private.

Differential privacy provides a well-founded means for protecting individual tuples in a table while releasing reasonably accurate aggregate properties of the entire table. It is robust against attackers with background knowledge about the database, the major weakness of access control type of protection. Achieving differential privacy usually requires perturbing statistics computed from the true database. The input privacy parameter ϵ (and δ if using the relaxed definition) are non-negative and are used to measure the degree of privacy protection. Smaller ϵ means better privacy as $\exp(\epsilon)$ is close to one.

2.3.1 Differentially private mechanisms

Differential privacy can be achieved by adding noise to the output of algorithms according to the privacy parameters (ϵ and δ) and the query *sensitivity*.

Global sensitivity and Laplace mechanism The global sensitivity of a query is the maximum possible difference in the output when evaluating the query on two neighboring graphs. E.g., the query asking for the size of a table has global sensitivity 1, because adding or removing one tuple changes the size by one. Let $Lap(b)$ be a Laplace random variable with mean 0 and scale b.

Definition 2.2 (Laplace mechanism [25])**.** Given query f on input x, the following algorithm $\mathcal{K}(f, x)$ is ϵ-differentially private:

$$\mathcal{K}(f, x) = f(x) + Lap(\mathsf{GS}_f/\epsilon)$$

where global sensitivity

$$\mathsf{GS}_f = \max_{\forall x_1, x_2 \ neighbors} |f(x_1) - f(x_2)|$$

An important property of differential privacy is that *post-processing* a noisy, differentially private output using any algorithm that does not access the original data cannot alter the privacy guarantee. Past research has shown that post-processing the noisy output can, however, have significant impact on utility. In addition, *composition rules* for differential privacy allow us to compute the ϵ privacy standard that results from the combined release of multiple query answers or releases. Precisely, if each release is ϵ_i-differential privacy, the combined is then $\sum_i \epsilon_i$-differential privacy.

Gaussian mechanism Correspondingly, a similar noise adding mechanism is available for the relaxed (ϵ, δ)-differential privacy. Slightly different, the global sensitivity is based on L_2 distance between output on two neighboring inputs and the noise is generated from Gaussian distribution. Let $Normal(\sigma)$ be a Normal random variable with mean 0 and scale σ.

Definition 2.3 (Gaussian Mechanism [23, 73])**.** Given query f on input x, the following algorithm $\mathcal{K}(f, x)$ is (ϵ, δ)-differentially private:

$$\mathcal{K}(f, x) = f(x) + Normal(\frac{\mathsf{GS}_f \sqrt{2\ln(2/\delta)}}{\epsilon})$$

where global sensitivity

$$\mathsf{GS}_f = \max_{\forall x_1, x_2 \ neighbors} ||f(x_1) - f(x_2)||_2$$

2.3.2 Differential privacy for graph data

The above definition of differential privacy applies on tabular data, where protection of individuals naturally nails down to each record in a database. When adapting differential privacy to graph data, such concept of "individual" is no longer obvious, because network contains not only major entities (nodes) but also their relationship

(edges). In tabular data, differential privacy relies on the precise definition of neighbors, adding or removing single record. In graph data, we can simulate the idea but with carefully selected "record".

If the neighbors are restricted to graphs by adding or removing single edge, we call it *edge differential privacy* [45]. Intuitively, edge differential privacy protects relationship among nodes in a graph, where the attacker cannot identify the existence of edges from output. Alternatively, *node differential privacy* [45] is another interpretation when the neighboring graphs consider inserting or deleting single node. Though node differential privacy is more desirable, as we can assume the "individuals" in a graph match better with nodes, it is usually infeasible in real application. This is because the global sensitivity associated with node differential privacy is usually too large for noise calibration. For instance, the number of edges in a graph, can change from n to 0 in node differential privacy if there is a single node connects every others. A compromise for that is *k-edge differential piracy* [45] where a group of k edges are protected and usually mechanisms designed for edge differential privacy can be easily adjusted to it. In Chapter 5, we follow the edge differential privacy, as it is the most common interpretations in the literature [45, 55, 83, 90, 109].

CHAPTER 3

AUDITING A DATABASE WITH RETENTION RESTRICTIONS

This chapter describes a framework for auditing the changes to a database system, which is sanitized in the presence of retention restrictions. We consider a historical data model and propose three kinds of rules for selectively obscuring or preserving sensitive data from the record of the past. We then address the problem of answering audit query on the incomplete data.

3.1 Introduction

Auditing the changes to a database is critical for identifying malicious behavior, maintaining data quality, and improving system performance. But an accurate audit log is an historical record of the past that can also pose a serious threat to privacy. In many domains, retention policies govern how long data can be preserved by an institution. Regulations mandate the disposal of past data and require strict retention periods to be observed. For example, the Fair Credit Reporting Act limits the retention, by credit reporting agencies, of personal financial records. In addition, institutions and companies often adopt their own policies limiting retention, choosing to remove sensitive data after a period of time to avoid its unintended release, or to avoid disclosure that could be forced by subpeona. Failure to dispose of the expired data can result in serious consequences and is often viewed as an institutional risk [112]. At the same time, other forces may require the preservation of records, for example when ongoing litigation makes removal of data unlawful. Institutions are

increasingly recognizing that a consistently enforced retention policy reduces their legal risk by ensuring that electronic data is handled properly [62].

Limited retention conflicts with the goals of accurate auditing, analysis, and prediction based on past history. This conflict is evident in the guidelines for record keeping published by a records management trade group [3], which include principles of data availability and data retention along with data disposal. Data owners thus have to carefully balance the need for accurate auditing with the privacy goals of retention policies. An emerging industry has begun to address the needs of these institutions, building systems that offer varying combinations of records and document management, archiving, eDiscovery, retention, and compliance services [26, 39, 80, 84, 91, 118, 119] . Unfortunately, existing mechanisms for auditing and managing historical records have few capabilities for managing the balance between these two objectives. Obeying a retention policy often means the wholesale destruction of the audit log.

In this chapter we propose a framework for auditing the changes to a database system in the presence of retention restrictions. We consider an historical data model and propose two kinds of rules for selectively removing or obscuring sensitive data from the record of the past. Despite the removal of information, it is often still possible for an auditor to monitor the record of actions taken on the database.

3.1.1 Applications

The tension between audit analyses and retention restrictions is present in a broad range of industries where sensitive records are managed, including financial services, healthcare, insurance, technology, education, telecommunications, and others. For example, financial legislation mandates limited retention periods for personal credit reports, including special treatment of negative credit events which are purged from records separately from other events. Search engines are not governed by legislation

in the U.S. but many elect to sanitize their search logs after 9 months. Logs are generally not disposed of completely, but certain fields are removed to reduce identifying information and to resist subpoenas and court orders.

Healthcare databases store sensitive information about patients, physicians, test results, diagnoses, billing details, and hospital procedures. State and federal laws specify record retention time frames that may depend on whether a patient is enrolled in medicare or medicaid, whether the patient is a minor, whether the medical procedure involves immunization, or on the statute of limitations for medical malpractice claims. At the same time, after mandated retention periods have passed, physicians may have discretion about how or when to dispose of records, or whether to partially sanitize records to remove personally identifiable data or sensitive diagnoses, while still permitting historical analysis. For example, we will define the operation of redaction on fields in a record. This could be applied to the diagnosis field of medical records while still permitting an analysis of a physician's consistency of diagnosis based on test results.

As another example, the office of information technology in a university is responsible for maintaining and monitoring network services for faculty, students, and staff. Network logs may contain information about machines, network connections, web browsing history, search engine requests, and/or file transfers, where users are identified by IP address or login name. Internal auditing may include analyzing logs for evidence of security vulnerabilities with the university. Researchers within the university may wish to perform traffic analysis on network logs. Lastly, external authorities such as the RIAA may request log data pertaining to specified users or specified content. In this setting, retention restrictions arise from privacy protections of individuals using the network. Some logs that are retained for network security purposes may be subject to removal of identifiers or sensitive content like web addresses. In addition, information technology staff reportedly prefer the timely removal

of some network logs so that they do not have to bear the cost of inquiries by external authorities.

Next we provide an overview of the motivation and contributions of this work through the following detailed example over a simple employment database. The schema and queries serve as a running example in later sections.

3.1.2 Example Scenario

We begin with a database consisting of tables belonging to a *client schema*. *Clients* interact with the database by submitting queries and updates, always on the current snapshot. In the running example used throughout this chapter, the client schema consists of a single table, S, describing employees:

$$S(\underline{eid}, name, department, salary)$$

The *auditor* is responsible for monitoring access to the database and tracking down malicious actions after they have occurred. Auditors typically inquire about *what* happened to the database, *when* it happened, and *who* did it.[1] To enable the auditor to query the state of the database over time, the system maintains an audit log table, L_S, for each table S in the client schema. Each modifying operation, issued by a client on S, is recorded in L_S along with additional *audit fields* describing the **time** of modification, the **type** of modification (insert, update, delete), and any other fields possibly of interest to the auditor. Table 3.1 shows an audit log table including audit fields recording the name of the issuing **client** and their **IP** address.

The audit log can easily be converted to an alternative transaction-time representation. Table 3.2 shows such a table, denoted T_S. It represents the complete data

[1] We are concerned here with auditing *modifications* only. We do not audit queries that read from the database.

history of the table, recording, in the **from** and **to** columns, the active period of each tuple in the database. Throughout the chapter we will use both the log-based and transaction-time representations as they each have benefits for expressing queries and defining concepts.

These historical tables can support a variety of queries of interest to the auditor. Some simple examples include:

A1. *Return all employees who earned a salary of 10 at some point in time.*

A2. *Return the clients who updated Bob's salary, and the time of update.*

A3. *Return the clients who updated any employee's dept, and the time of update.*

A4. *Return the time periods when Bob earns a salary of 10.*

Some audit queries are conventional queries over a transaction-time data model (such as A1, A4). Others ask specifically about changes, and reference the special audit fields contained in the audit log (such as A2, A3).

The *compliance officer* is a trusted entity, responsible for enforcing data retention restrictions arising from privacy regulations or institutional policies. These policies are typically non-negotiable – they must be respected by all users of the system, including the auditor. We propose two kinds of declarative retention rules for limiting the lifetime of data. The compliance officer is also responsible for enforcing preservation rules, which reflect requirements to keep certain data items in the database. Notably, these policies are expressed in terms of T_S, the transaction time table describing the data history. This is the most natural choice because retention policies refer only to the client schema, and to the notion of time.

Our first retention rule is called **redaction**. When redaction is applied to an attribute value, it removes the value but does not hide its existence. For example, a redaction rule may say: *Hide Bob's salary between time 0 and 250.* The second

client	**IP**	**time**	**type**	`eid`	`name`	`dept`	`sal`
Jack	1.1.1	0	ins	101	Bob	Sales	10
Jack	2.1.1	100	upd	101	-	-	12
Kate	3.1.1	200	upd	101	-	Mgmt	-
Kate	4.1.1	300	upd	101	-	-	15
Jack	1.1.1	0	ins	201	Chris	HR	8
Jack	2.1.1	300	upd	201	-	Mgmt	10
Kate	4.1.1	500	del	201	-	-	-

Table 3.1: The audit log L_S describing the history of operations performed on a client table with schema *S(eid,name,dept,salary)*. Columns **client** and **IP** are audit fields.

`eid`	`name`	`dept`	`sal`	**from**	**to**
101	Bob	Sales	10	0	100
101	Bob	Sales	12	100	200
101	Bob	Mgmt	12	200	300
101	Bob	Mgmt	15	300	now
201	Chris	HR	8	0	300
201	Chris	Mgmt	10	300	500

Table 3.2: The transaction-time table T_S describing the data history of the client table. It is derived from the audit log in Table 3.1.

operation, called **expunction**, is more extreme. When a tuple is expunged, it is completely removed, along with all evidence of its existence. For example, an expunction rule may say: *Remove the record of all employees in the HR department between time 0 and 300*. We believe these rules are sufficiently expressive for practical applications, allowing users to selectively choose related data items, which could be tuples, selected tuples, or individual attribute values [76]. We also support a basic rule for **preservation**, which takes priority over the removal rules above, ensuring that specified records are not altered or removed.

eid	name	dept	sal	from	to
101	Bob	Sales	**sx**	0	100
101	Bob	Sales	**sy**	100	200
101	Bob	Mgmt	**sy**	200	250
101	Bob	Mgmt	12	250	300
101	Bob	Mgmt	15	300	now
201	Chris	HR	8	0	300
201	Chris	Mgmt	10	300	500

Table 3.3: The transaction time table, transformed under the following retention policies: $\text{Redact}_S(name = Bob, \{salary\}, [0, 250])$ and $\text{Expunge}_S(dept = HR, [0, 300])$. (The gray row has been deleted.)

Applying a set of retention rules transforms the stored history of the database.[2] Table 3.3 shows a new transaction-time table, the result of applying the retention rules to the table T_S. In applying the redaction rule, salary values have been replaced with variables (**sx, sy**). Instead of suppression with NULLs, we use variables to support more accurate auditing by retaining more information, as different values are suppressed to different variables. Also note that there is an extra row in Table 3.3 because the time interval [200,300] in the original data has been split into two intervals: [200,250], in which Bob's salary is hidden, and [250,300], in which Bob's salary can be revealed to be 12. In applying the expunction rule, Chris's membership in the HR department has been removed from the history: he is now only in the Mgmt department from time 300 to 500. For illustration purposes, the expunged row is included in Table 3.3, but displayed with a gray background.

A main goal of this chapter is provide a proper semantics for audit queries in the presence of retention policies. Because the transformed history has tuples removed by expunction and values obscured by redaction, the answers to audit queries may

[2]As a practical matter, retention rules may be applied physically, altering storage of the table, or logically, in which access is restricted but hidden data is still physically stored. Sec. 3.8 provides further detail.

be uncertain or, in some cases, provide false information. We reconsider the previous audit queries under retention restrictions:

A1. *Return all employees who earned a salary of 10 at some point in time.*

This query is a straightforward selection on the transaction-time table. On the original data in Table 3.2 the answer to this query is {Bob, Chris}. On Table 3.3, under the retention policy, the answer to this query includes Chris as a *certain* answer. However, Bob is only a *possible* answer because the predicate depends on the unknown value of variables **sx** and **sy**. Our implemented system returns both answers, labeled appropriately as possible or certain.

A2. *Return the clients who updated Bob's salary, and the time of update.*

The answer to this query on the original data is {(Jack, 100), (Kate, 300)}. The transformed history in Table 3.3 shows that Bob's salary definitely changed at time 100 (from **sx** to **sy**) and at time 300 (from 12 to 15). In addition, it *may* have changed at time 250 (from **sy** to 12), depending on the unknown value of variable **sy**. (Note that the uncertainty about this change is crucial – if it is possible to deduce that the change did not occur, then it is clear that Bob's salary was indeed 12 between 250 and 300, and the retention policy is violated.)

In order to fully answer the query, we must use the audit log to get the names of the clients who issued the update. Jack and Kate performed the updates at time 100 and 300, respectively, so the certain answers to this query are: {(Jack, 100), (Kate, 300)}. A subtlety here is how to return the possible answer for the update at 250, since there is no known client that performed that update. The possible answer that could be returned is: (NULL,250), but not if it reveals that this is a fake update.

A3. *Return the clients who updated any employee's dept, and the time of update.*

The answer to this query on the original data is {(Kate, 200), (Jack, 300)}, which can easily be computed from the original audit log L_S. In the transformed history in Table 3.3 we find evidence of only one update to the department field, at time 200. This is a result of the expunction policy that removed Chris' record from time 0 to 300. Thus, the answer to this query under the retention policy is {(Kate, 200)} and the record of Jack's update is lost.

Notice that the answer to query A3 is incorrect: a tuple that is in the true answer (i.e. with respect to the original data) is omitted from the new answer. From the auditor's perspective this is a worse outcome than that of A1 and A2 where the true answer is one of the possible answers. One of the goals of our framework is to provide answers to audit queries that, while possibly imprecise, do not lead to false conclusions. Also note that in reasoning about the answers to queries A2 and A3 we referred to the transformed transaction-time table and used it to infer actions that were performed on the database. Later in the chapter we make this process explicit by computing a **sanitized audit log**, consistent with the retention policies, that can be queried directly.

The answer to an audit query under retention rules usually consists of two parts: certain tuples and possible tuples. Such results are *uncertain* answers, because they are computed on a history with incompleteness introduced by applying retention rules. On the contrary, querying the original history without retention rules returns a *real* answer. Intuitively, each uncertain answer represents a set of real answers, each of which is returned by the query over some original history that is consistent with the transformed history under the retention policies. In the case of A1, the uncertain result could represent two real answers. One is {Chris} when A1 is executed over the history where neither sx nor sy is 10 and the other is {Chris,Bob} when either sx or sy is 10. Similarly, if the uncertain result of A1 contains two possible tuples, say, Bob and Ann, we have four real answers represented, {Chris}, {Chris,

Bob},{Chris, Ann} and {Chris, Bob, Ann}. Here we simply assume the existence of each possible tuple in a real answer is *independent* of others, and thus we have four different ways of choosing two possible tuples. We propose the **Tuple-Independent model (TI)** for answering audit queries under retention policies, and define the semantics of uncertain answers returned by TI under this independence assumption. Later we will show, due to this assumption and the extended relational algebra, that there is no extra cost to decide certain and possible tuples of the query results since they are efficiently computed during query evaluation. Thus TI guarantees efficiency and this fact serves as a major advantage of the TI model, along with it simplicity. However, the independence assumption is not always correct and thus the information delivered by the uncertain answer is not precise, as demonstrated by the following query A4. To solve this problem, we introduce a more sophisticated model, the **Tuple-Correlated model (TC)**, which does not rely on an independence assumption and gives precise interpretations of uncertain answers.

A4. *Return the time periods when Bob earns a salary of 10.*

The answer to this query on the original data is {(0,100)}, which can easily be computed from the original audit log L_S. In the transformed history in Table 3.3, our result is {(0,100),(100,200),(200,250)} and all are possible answers due to the unknown value of two variables **sx** and **sy**. When using TI, we assume the three periods are independent of each other and therefore we could interpret them as eight different real answers. However, a closer look will tell us such an assumption is invalid. (100,200) and (200,250) are correlated because they are bound to the same variable sy. (0,100) is also not independent of either (100,200) or (200,250) because sx and sy are correlated: they are not equal (recall that they represent distinct values). In fact, this uncertain result only represents three different real answers. If sx equals 10, then (0,100) is the output; if sy equals 10, then (100,200) and (200,250) is the output; finally

if neither sx nor sy is 10, then the output is empty. For a more accurate representation we use the TC model which can maintain the correlations among three time periods. Instead of indicating "certain" or "possible" directly for each tuple, TC records extra information in the form of conditions associated with each tuple. The example above may be represented as:

$$\begin{aligned} (0, 100): &\qquad sx = 10, \\ (100, 200): &\qquad sy = 10, \\ (200, 250): &\qquad sy = 10 \end{aligned}$$

Because all three equations are satisfiable, we have three possible tuples. Tuples (100,200) and (200,250) occur together, or not at all, depending on the assignment to sy.

Query A4 demonstrates that there are cases where the independence assumption fails and thus the TI model is incapable of representing the result accurately. Our TC model abandons the independence assumption and is able to provide accurate answers by recording equalities and inequalities of variables. We use this extra information to decide certain and possible tuples. From the auditors' perspective, the ability to calculate the correlation is important and delivers more valuable information. For instance, given a possible suspect Alice, if the auditor has some external information in Alice's favor, the TC model can help to answer questions like "Who remains a suspect if I assume Alice is not a suspect?" Our system (using either the TI or TC models) returns uncertain answers which reflect the unavoidable imprecision of carrying out an audit task in the presence of a partially removed history. In the absence of our techniques, a conventional system would be unlikely to produce valid query answers at all.

We use the term *expressiveness* to measure the ability to precisely represent the set of correct answers. TC is strictly more expressive than TI because it can interpret the uncertain answer of A4 but TI can't. After analyzing their expressiveness and investigating other alternatives later in this chapter, we conclude that the combination of TI and TC meet the needs of our application. We will see that the cost of TC's expressiveness is the decreased efficiency of deciding which tuples are certain and which are possible.

In summary, the main contributions of this chapter are:

- We propose declarative rules for expressing retention restrictions over an historical data model. (Section 3.4)

- We define the tuple-independent model (TI) for answering audit queries in the presence of retention restrictions and we analyze the impact of retention policies on the accuracy of audit queries. (Section 3.5)

- We present the tuple-correlated model (TC) for answering audit queries. Tuple level correlations are captured by additional conditions appended to each tuple. We define the extended relational algebra for TC. We compare the expressiveness of TI and TC and prove that TC is a complete data model meaning that it can represent any possible set of answers. (Section 3.6) We show the advantages of TI and TC in comparison to other models.

- We discuss the complexity of deciding whether tuples are possible or certain in Section 3.7. In TI, this is given explicitly by an extra column. In TC, deciding that a tuple is possible is NP-complete and deciding that it is certain is coNP-complete. However, for a large subclass of instances, we show that efficient scheduling algorithms can determine possiblility in P.

- We implement our framework via extensions to Postgres, showing that uncertain answers can be computed efficiently over both models. (Section 3.8)

- We demonstrate (through simulation on sample data) that useful auditing can be performed in the presence of retention restrictions, despite uncertain answers. The study of the impact of retention policies on the accuracy of query results under TI and TC shows cases where TC can significantly improve accuracy over TI. (Section 3.9)

We describe our threat model, in Section 3.2, and our data model and queries, in Section 3.3. We distinguish our contributions from related work in Section 3.10.

3.2 Threat Model & Security Objectives

3.2.1 Adversaries

Our threat model focuses on two major categories of adversary: auditors and external authorities.

An *auditor* is an authenticated user of the system who is permitted to ask queries about past events in the database. We use the single term *auditor* to refer to either an entity external to the enterprise who is authorized to perform audit tasks, or a user internal to the enterprise who wishes to compute analytics or monitor changes in the database. We assume auditors are not capable of subverting standard authentication procedures or access controls imposed by the compliance officer. In our framework, this means that the auditor is restricted to the sanitized data history only.

An *external authority* is an entity, such as a legislative body, a governmental institution, or a legal authority, capable of issuing audit queries that the enterprise is compelled to answer using all information available in the database. An external authority is not restricted by access controls imposed by the compliance officer. However, information that is physically removed from the data history will no longer be available to anyone, even the external authority. In addition, the external authority can issue a data hold to the compliance officer, preventing the compliance officer from removing specified data from the history.

3.2.2 Threats and Security Objectives

Data Disclosure The primary threat we address is unintended disclosure of the data history. When the compliance officer intends to protect portions of the data history through one or more retention policies, but that data history is nevertheless exposed to an auditor or external authority, then data disclosure has occurred. For example, in our motivating scenario, if Bob's salary is not appropriately sanitized by Policy 1, the answer to Query A1 may have Bob as a certain answer, resulting in a compromise of Bob's privacy.

Maintenance of data holds We assume that if an external authority issues a data hold for a portion of the data history, the enterprise is required to retain that history for later audit queries by the external authority. Failure to comply with this requirement may result in significant liability for the enterprise, so we consider maintenance of data holds an important security property of our framework.

We consider avoiding data disclosure and respecting data holds as non-negotiable requirements of our framework, treating these as hard constraints that must be met. Subject to these constraints, we desire to provide the best utility and availability possible for auditing. In the best case, audit answers are precise. If they are not precise, the auditor may be faced with uncertainty about the actual audit query answer, but we nevertheless insist that answers be *sound*, so that they do not lead to false conclusions. These assumptions favor compliance over auditing, and imply that some retention policies established by the compliance officer may not allow accurate auditing for some auditing queries. It is possible for the compliance officer to detect the interaction between a retention policy and audit query (see discussion in Sec. 3.4.3).

There are other enterprise security threats that are not the primary focus of this work. We assume that conventional methods are used to prohibit database clients from altering the data history or log in any manner other than through inserts,

updates, and deletes on the current snapshot. We also assume that the compliance officer is trusted to implement policies correctly: we do not defend against the threat posed by an untrusted compliance officer intentionally altering the log. Such threats have received considerable attention elsewhere [28, 43, 44, 80, 91, 96, 104]. Lastly, while external authorities are capable of accessing any data stored by the enterprise, we assume they cannot carry out a full forensic examination of the enterprise system to reveal further data remnants that may be retained. Such threats have been considered by prior work [105] and we assume suitable countermeasures are employed.

3.2.3 Achieving security objectives

The retention policies described in this chapter have a single well-defined semantics (described in Sec 3.4). But applying the retention policies to the data history can be done in one of two ways: physically or logically. Logical implementation addresses the threat of data disclosure only with respect to auditors, but not external authorities. An advantage of logical implementation is that it is easy to modify or re-apply the retention policy, and because data is not physically removed, logical application never conflicts with data hold requirements. Physical implementation, in which redaction and expunction result in physical removal of data, is more secure, addressing the threat of data disclosure for both auditors and external authorities.

3.3 Data Model and Audit Queries

In this section we describe our data model, based on backlog and transaction-time databases [52, 53], and our language for expressing audit queries.

3.3.1 Data model

Let $(S_1, \ldots, S_k)$ be the client schema. We refer to each relation S_i as a *regular* relation to distinguish it from transaction-time relations defined below. $\mathit{tuples}(S_i)$ is

the set of all tuples that could occur in S_i (i.e., the cross-product of the attribute domains).

Audit Log

An audit log is a complete record of the operations on a client table over time, and we maintain an audit log table L_S for each table S of the client schema. Each row in L_S represents a transaction modifying a tuple of S. Table 3.1 shows an example audit log table. In general, the schema of L_S is:

$$(\langle \textit{audit-fields} \rangle,\ \textit{ttime},\ \textit{type},\ \langle \textit{client-fields-from-}S \rangle)$$

The *audit fields* may contain an arbitrary set of attributes describing facts about the transaction. In our examples, the audit fields record the name of the issuing **client** and their **IP** address, but in general they may include many other fields describing the context of the operation. *ttime* is a time stamp, from a totally-ordered time domain $\mathcal{T}$, reflecting the commit time of the transaction. We assume each transaction receives a unique time stamp. The *type* field describes the modification as an insert, update, or delete. The fields of the client schema describe the changes in data values. If the transaction is an *insert*, each attribute value is included; for updates, only modified values are included, with unchanged attributes set to NULL; for deletes, all attribute values are NULL. This description of an audit log is essentially a backlog database [53] with the addition of audit fields.

We assume that each audit record refers to a unique tuple, identified by the key of the client table. In practice, a transaction may affect multiple tuples. If necessary, this relationship can be recorded in a statement-id, relating the changes to tuples made by a statement. Without loss of generality we omit this.

Transaction-time relation

A *transaction-time relation* (a *t-relation* for short) represents the sequence of states of a relation in the client schema. Formally, a t-relation over S is a subset of $tuples(S) \times \mathcal{T}$. A tuple $(p_1, \ldots, p_n, t) \in T_S$ represents the fact that tuple $(p_1, \ldots, p_n)$ is active at time instant t. In examples (and our implementation) we use the common representation for t-relations in which $(p_1, \ldots, p_n, from, \rightarrow)$ means that $(p_1, \ldots, p_n)$ holds at each instant t, for $from \leq t \leq \rightarrow$. Table 3.2 is an example of a t-relation.

Audit log versus T-relation

Given an audit log table L_S, a unique t-relation can be computed from it in a straightforward way by executing each statement. After a modification, the values of a tuple are active until the time instant of the next operation modifying that tuple. We use *exec* to indicate this procedure, and we define T_S to be $exec(L_S)$ for each S in the client schema.

It is also possible to reverse this procedure, computing an audit log from a t-relation (although no audit fields will be included). This procedure, denoted $exec^{-1}$, computes initial insertion transactions at the time instant a new tuple is created, subsequent update transactions at the instant of each change to a tuple, and (for tuples that are no longer active) deletion transactions. Notice that computing an audit log from T_S will reproduce a table similar to L_S but with the audit fields removed: $\Pi_{ttime,type,S}(L_S) = exec^{-1}(T_S)$.

The audit log L_S and the t-relation T_S represent similar information. As a practical matter it is not necessary to maintain both. However, in the formal development presented here, each representation serves an important purpose. We will see in the next section that retention policies are defined in terms of T_S, and can be applied directly to T_S. But T_S does not include audit fields. We will also reconstruct an au-

dit log from the protected T_S in order to make explicit the possible inferences about changes to the database.

3.3.2 Audit queries

A variety of interesting audit queries can be expressed over T_S and L_S. L_S is a regular relation, but queries over t-relation T_S may use extended relation algebra operators to cope with transaction-time. We omit a formal description of these operators, which can be found in the literature [17, 30], and instead present examples highlighting their features.

The example audit queries from Section 3.1.2 are expressed as follows on T_S or L_S:

A1. *Return all employees who earned a salary of 10 at some point in time.* $\Pi_{name}(\sigma_{sal=10}(T_S))$

A2. *Return the clients who updated Bob's salary, and the time of update.*

$$\Pi_{client,ttime}(\sigma_{type=upd \wedge name=Bob \wedge sal \neq NULL}(L_S))$$

A3. *Return the clients who updated any employee's dept, and the time of update.*

$$\Pi_{client,ttime}(\sigma_{type=upd \wedge dept \neq NULL}(L_S))$$

A4. *Return all the time periods when Bob earns a salary of 10.* $\Pi_{from,to}(\sigma_{sal=10}(T_S))$

Conventional joins on t-relations are possible, as well as joins between a t-relation and regular relation. For example, our audit log L_S can be joined with T_S on the *ttime* attribute. In addition, we can use *concurrent cross-product* (denoted $\times^{\diamond}$) or *concurrent join* (denoted $\bowtie^{\diamond}$) as binary operators on t-relations that combine tuples active at common time periods. The following additional example query includes a concurrent self join on T_S:

client	IP	ttime	type	eid	name	dept	sal
Jack	1.1.1	0	ins	101	Bob	Sales	**sx**
Jack	2.1.1	100	upd	101	-	-	**sy**
Kate	3.1.1	200	upd	101	-	Mgmt	-
NULL	NULL	250	upd	101	-	-	12
Kate	4.1.1	300	upd	101	-	-	15
NULL	NULL	300	ins	201	Chris	Mgmt	10
Kate	4.1.1	500	del	201	-	-	-

Table 3.4: A sanitized audit log, $P(L_S)$, transformed under the retention policies of Section 3.1.2 and Example 3.5.

A5. *Return all employees who worked in the same department as Bob at the same time.*

$$\Pi_{name}(\sigma_{name'=Bob}(T_S \bowtie^{\diamond}_{dept=dept'} T'_S))$$

Finally, the *time-slice* operator restricts a t-relation to a specified interval in time. For the interval $[m, n]$, it can be defined as: $\tau_{m..n}(R) = R \times^{\diamond} \{\langle m, n \rangle\}$ where $\{\langle m, n \rangle\}$ is a singleton t-relation without user-defined attributes. The result of applying the time-slice operator is a t-relation. A regular relation representing the snapshot database at time m can be written as $\pi_{S-\{\text{from,to}\}}\ (\tau_{m..m}(T_S))$.

3.4 Describing and Applying Retention Policies

In this section, we define the semantics of our redaction, expunction, and preservation rules, and discuss how they are applied to the stored history. When the implementation respects the semantics of these rules, the threats and security properties in Sec. 3.2 will be satisfied.

3.4.1 Retention policy definitions

Retention policies are used to restrict access to tuples or attribute values in one or more historical states of the database. The need for retention policies arises from the sensitivity of data items in the client schema. Thus it is most natural to express

retention policies in terms of the t-relation, T_S, which describes states of the client relation as it evolves through time. We define our retention policies formally below as transformations on T_S.

Our first retention operation is called **redaction**. It suppresses attribute values in tuples for a specified time period. Redaction is useful because it hides sensitive data values, but preserves the history of modification of the tuple. Our second retention operation is called **expunction**. An expunged tuple is removed from history, and the historical record is modified accordingly to hide its existence.

These two operators serve different purposes as they enact *value* removal in the case of redaction, and *existence* removal in the other. Expunction is a more extreme operation because it does not merely suppress information, but changes the historical record in ways that can substantially change answers to audit queries. We believe that a variety of privacy policies can be satisfied through the use of redaction policies alone, which will lead to more accurate auditing.

In the definitions that follow, a Boolean condition ϕ, on client relation S, is a Boolean combination of comparisons $S.A\,\theta\,c$, or $S.A\,\theta\,S.B$, for any $\theta \in \{=, \neq, <, \leq, >, \geq\}$.

Definition 3.1 (Expunction Rule)**.** An expunction rule, over a client table S, is denoted $\mathcal{E} = \text{Expunge}_S(\phi, [u, v])$ where ϕ is a Boolean condition on attributes of S, and $[u, v]$ is a time interval ($u, v \in \mathcal{T}$, and $u \leq v$).

An expunction rule asserts that all tuples matching condition ϕ should be removed from a specified interval in time. When an expunction rule E is applied to a t-relation T_S, the intended result is a new t-relation. Denoted $\mathcal{E}(T_S)$, this new t-relation consists of all facts from T_S *except* those that satisfy ϕ and have time field in $[u, v]$:

Definition 3.2 (Expunction Rule Application)**.** For a client relation S, let T_S be a t-relation over S, and

$$\mathcal{E} = \text{Expunge}_S(\phi, [u, v])$$

be an expunction rule. The application of $\mathcal{E}$ to T_S, denoted $\mathcal{E}(T_S)$, is a new t-relation with the same schema: $\mathcal{E}(T_S) = T_S - \{x \in T_S \mid \phi(x) \wedge x.t \in [u, v]\}$

Unlike expunction, a redaction rule does not remove tuples from the historical record. Instead, a redaction rule asserts that the values of certain attributes should be suppressed in all tuples that match condition ϕ and are active during a specified time interval.

Definition 3.3 (Redaction Rule)**.** A redaction rule, over client table S, is denoted $\mathcal{R} = \text{Redact}_S(\phi, \mathbf{A}, [u, v])$ where ϕ is a Boolean condition on attributes of S, $\mathbf{A}$ is a subset of the columns in S, and $[u, v]$ is a time interval ($u, v \in \mathcal{T}$, and $u \leq v$).

When a redaction rule $\mathcal{R}$ is applied to a t-relation T_S, the intended result is a new t-relation, denoted $\mathcal{R}(T_S)$, in which some attribute values have been suppressed. To formalize $\mathcal{R}(T_S)$ we use a suppression function $\text{supp}(x, \mathbf{A})$ which replaces attributes of $\mathbf{A}$ in the transaction-time tuple x with variables. For example, if $x = (101, \mathit{Bob}, \mathit{Sales}, 10, 300)$ then $\text{supp}(x, \{\mathit{dept}, \mathit{salary}\}) = (101, \text{Bob}, \mathbf{dx}, \mathbf{sx}, 300)$. We assume that suppressions of distinct values always use distinct variable names, and that all instances of a value are replaced by the same variable.

Definition 3.4 (Redaction Rule Application)**.** For a client relation S, let T_S be a t-relation over S, and $\mathcal{R} = \text{Redact}_S(\phi, \mathbf{A}, [s, t])$ be a redaction rule. The application of $\mathcal{R}$ to T_S, denoted $\mathcal{R}(T_S)$, is a new t-relation with the same schema:

$$\begin{aligned} \mathcal{R}(T_S) \quad = \quad & \{\text{supp}(x, \mathbf{A}) \mid x \in T_S, \phi(x), x.t \in [u, v]\} \cup \\ & \{x \mid x \in T_S, \neg\phi(x) \vee x.t \notin [u, v]\} \end{aligned}$$

We assume for simplicity that $\mathbf{A}$ does not contain the key for table S. If the key for R is sensitive, and subject to retention policies, a surrogate non-sensitive key attribute can be introduced to the schema. This means that even if all attributes of the schema are redacted, the history of changes to a tuple is still preserved.

Having applied a redaction policy, the resulting table $\mathcal{R}(T_S)$ is formally an *incomplete* t-relation. It is a representation of a set of possible worlds, each resulting from a different substitution of distinct values for the variables introduced by the suppression of attributes. We define incomplete relations formally in Section 3.5.

Retention policy composition

Retention rules can be combined to form composite retention policies. A set of redaction rules is combined by hiding any attribute value that satisfies the selection condition and time-period of *any* individual redaction rule. A set of expunction rules is combined by removing all tuples satisfying *any* individual expunction rule. Expunction rules take precedence over redaction rules: a tuple satisfying both an expunction and redaction rule will be removed rather than suppressed.

Example 3.5. In Section 3.1.2, we described informally two retention policies. The redaction rule that *hides Bob's salary between time 0 and 250* is written formally as $R = \text{Redact}_S(\text{ name='Bob', } \texttt{sal}, [0, 250])$. The expunction rule that *removes the record of all employees in the HR department between time 0 and 300* is written $E = \text{Expunge}_S(\text{dept='HR', } [0, 300])$. Table 3.3 is the t-relation that results from applying both E and R to the original table T_S shown in Table 3.2.

Suppression by Variables vs. NULLs

The choice to use variables instead of NULL values for cell suppression allows for improved audit accuracy but can sacrifice confidentiality because it reveals when two redacted values are identical. For example, suppose Bob's salary was 10 at time x but is later redacted. If Bob has the right to access both his and other employees'

information, he may find Jack's salary at time y is equal to his redacted salary at time x, allowing him to infer that Jack has salary 10 at time y, in violation of the redaction policy.

Nevertheless, we believe this is a worthwhile trade-off and we show in Section 3.9 that the use of variables can substantially increase auditing accuracy for some queries. Our framework can easily be adapted to a suppression function using NULL values.

3.4.2 Sanitizing the audit log

Consider a policy $\mathcal{P}$ consisting of redaction and expunction rules. According to the definitions above, we apply the policy to T_S to get the t-relation $\mathcal{P}(T_S)$. As we have seen in the examples of Section 3.1.2, the answers to audit queries are not determined completely by the table $\mathcal{P}(T_S)$. For one, the audit fields in L_S are not present. We must use L_S in combination with $\mathcal{P}(T_S)$ to answer queries that reference the audit fields. In addition, the operations applied to the database need to be inferred from $\mathcal{P}(T_S)$ which represents just the history of database states. In order to combine audit field information, and to make explicit the changes to the database that are implied by $\mathcal{P}(T_S)$, we compute a *sanitized log* consistent with $\mathcal{P}(T_S)$. This new log is denoted $\mathcal{P}(L_S)$ and has the property that running it results in $\mathcal{P}(T_S)$, that is: $exec(\mathcal{P}(L_S)) = \mathcal{P}(T_S)$. The auditor, and other users, will have access to both $\mathcal{P}(T_S)$ and the sanitized audit log. Together we refer to these as the sanitized history. The relationship between the audit log and transaction-time tables in our framework is illustrated in Figure 3.1.

When computing the sanitized history, we hope to satisfy the following properties.

- A sanitized history is **secret** if it respects the semantics of the policy, hiding tuples and values appropriately. This means it is not possible to infer from the protected history anything that is not present in $\mathcal{P}(T_S)$ (the defined meaning).

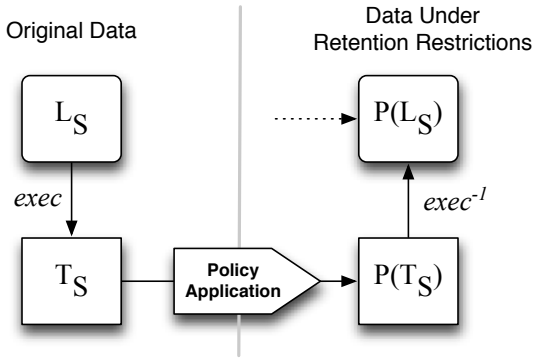


Figure 3.1: Illustration of the relationships between original history (L_S and T_S) and the history under retention policy P. $P(T_S)$ is defined directly, while $P(L_S)$ is the sanitized log derived from $P(T_S)$ and including audit fields from L_S.

This property defines the fundamentals of preventing data disclosure (in Section 2).

- A sanitized history is **sound** if it omits information, but does not lead to false answers to audit queries. This property is ensured for all queries if the possible worlds implied by $\mathcal{P}(T_S)$ include the original history. In that case, the true answer to any audit query must be a possible answer under retention restrictions. This property is essential for data utility (in Section 2) and it provides the basis for answering queries precisely.

Note that for any redaction rule $\mathcal{R}$ and expunction rule $\mathcal{E}$, $\mathcal{R}(T_S)$ and $\mathcal{E}(T_S)$ are secret by definition. The challenge to secrecy comes from integrating L_S. Also note that expunction policies necessarily violate soundness. Because an expunction policy changes history by removing records, it produces false answers to audit queries.

Definition 3.6 (Sanitized Log)**.** Let $\mathcal{P}$ be a retention policy consisting of redaction rules, expunction rules, or both, and let $\mathcal{P}(T_S)$ be the (possibly incomplete) t-relation that results from applying $\mathcal{P}$ to T_S. The sanitized log under $\mathcal{P}$ is denoted $\mathcal{P}(L_S)$ and is defined as follows:

1. Treating any variables present in $\mathcal{P}(T_S)$ as concrete data values, compute the audit log table $exec^{-1}(\mathcal{P}(T_S))$

2. Let $L_S^0 = \Pi_{\langle audit\text{-}fields \rangle, ttime}(L_S)$

3. $\mathcal{P}(L_S) = L_S^0 \bowtie\!=_{ttime} exec^{-1}(\mathcal{P}(T_S))$

This procedure first uses the $exec^{-1}$ to compute an audit log from $\mathcal{P}(T_S)$. Then we extract the audit fields and time column from the original audit log. This table, L_S^0, is then joined with $exec^{-1}(\mathcal{P}(T_S))$. We use a right outer join to preserve tuples in $exec^{-1}(\mathcal{P}(T_S))$ which may not have a match in L_S^0. This occurs when the application of a redaction policy splits the active interval of one or more records. It suggests that an update operation occurred in the history, but the time instant of this update does not match any update in the original audit log.

Example 3.7. Table 3.4 is the sanitized audit log computed according to the above definition, for the policy described in Example 3.5.

Note that Definition 3.6 is not itself an attractive strategy for computing the sanitized log. We describe our implementation of policy application in Section 3.8. In addition, we will see below that policies can be "applied" logically, in which case $\mathcal{P}(L_S)$ need not be materialized.

3.4.3 Retention policy analysis

We can show the following properties of the sanitized log.

Proposition 3.1. *Let L_S be an audit log, T_S the t-relation derived from it, and let $\mathcal{P}$ be a retention policy consisting of a set of redaction rules $R_1 \ldots R_n$ where each $R_i =$ $Redact_S(\ \phi_i,\ \mathbf{A}_i,\ [u_i, v_i])$.*

- *The computation of $\mathcal{P}(L_S)$ is* sound.

- *The computation of* $\mathcal{P}(L_S)$ *is* secret *iff*
  $u_i, v_i \in \Pi_{ttime}(L_S)$ *for all* i.

Proof. (Sketch) Soundness follows from that fact that $\mathcal{P}(T_S)$ is sound, and the fact that $\mathcal{P}(L_S)$ is consistent with $\mathcal{P}(T_S)$, in the sense that $exec(\mathcal{P}(L_S)) = \mathcal{P}(T_S)$. It follows that the original history is one possible world of $\mathcal{P}(L_S)$. If the condition $u_i, v_i \in \Pi_{ttime}(L_S)$ fails, then there are dangling tuples in the join described in Definition 3.6. The absence of audit fields leaks information and violates secrecy. If the condition holds then there are no dangling tuples. Secrecy follows from the fact that $\mathcal{R}(L_S)$ is consistent with $\mathcal{R}(T_S)$ and uses only the projection, L_S^0, of L_S. □

The sanitized log from Example 3.7 and Table 3.4 demonstrate the problems that result from arbitrary redaction intervals. These policies split intervals and suggest phantom updates that cannot be convincingly represented in the log. The failure of secrecy appears not to be merely an artifact of the semantics of redaction, but instead a fundamental difficulty in presenting an audit log that is consistent with a redacted data history. It is possible that secrecy could be achieved by introducing additional uncertainty about phantom modifications, but this entails a more powerful model of incompleteness, potentially sacrificing efficiency, and degrading audit query accuracy. Further investigation is a topic of future work.

As a practical matter, to avoid sacrificing secrecy for redaction rules, the desired time interval $[u, v]$ of each redaction rule can be shifted, either forward or backward, to the time of the nearest modification (to any field) in the log.

Policy/Query Independence

It is possible to decide statically, for a given policy and audit query, whether the query answer will be unaffected by the policy. This problem is closely related to the study of view independence of updates [10, 11]. Here the audit query occupies the

place of the view. Our retention policies can be considered deletions (in the case of expunction) or updates (in the case of redaction). Known results provide sufficient conditions for determining policy-query independence in our framework.

3.4.4 Supporting preservation rules

Redaction and expunction are removal rules. They implicitly indicate the semantics of data holds: the system only removes information that satisfies removal rules, and retains the rest. Thus, when there is a litigation hold, we change the existing removal rules accordingly to prevent unwanted deletion. However, a specific preservation rule may provide more flexibility for compliance officers. Our framework is able to support tuple-level preservation rules. A preservation rule tells the system to retain all tuples matching the conditions in ψ for a specified interval of time.

Definition 3.8 (Preservation Rule)**.** A preservation rule, over a client table S, is denoted $\mathcal{H} = \text{Presrv}_S(\psi, [u, v])$ where ψ is a Boolean condition on attributes of S, and $[u, v]$ is a time interval ($u, v \in \mathcal{T}$, and $u \leq v$).

When a preservation rule alone is applied to a t-relation, the t-relation is unchanged. When a preservation rule and a removal rule are applied together, the process of generating the new t-relation should ensure that those tuples matching the preservation rule are always retained, taking priority over the removal rule. To maintain data holds (described in Section 2), preservation rules must be applied correctly.

Definition 3.9 (Preservation Rule Application)**.** For a client relation S, let T_S be a t-relation over S, and $\mathcal{P}$ be the current set of removal rules (expunction and redaction). If a preservation rule $\mathcal{H} = \text{Presrv}_S(\psi, [u, v])$ is added to $\mathcal{P}$ to get $\mathcal{P}_+ = \mathcal{P} \cup \{\mathcal{H}\}$, then we generate a new set of removal rules, $\mathcal{P}'$, from $\mathcal{P}$ by transforming each condition $p.\phi$ for $p \in \mathcal{P}$ into $p.\phi \wedge (\neg\psi \vee (p.t \notin [u, v]))$ where $p.t$ is the specified time period in original rule p. If $\mathcal{P}_+(T_S)$ denotes the application of all the rules, we then have: $\mathcal{P}_+(T_S) = \mathcal{P}'(T_S)$

The definition above defines the semantics of integrating preservation rules by logically transforming the original removal rules. It follows from the definition that preservation rules take precedence over removal rules. Further, the properties of removal rules and sanitization processes defined earlier in this section hold also for policies that include preservation rules.

3.4.5 Physical vs. logical policy application

The discussion above has implicitly suggested the physical application of retention policies to the audit log and derived transaction-time table, in which record removal and attribute suppression are reflected in the storage system. Physical sanitization is appropriate when it is necessary to defend against external authorities as well as auditors, or when privacy policies mandate direct removal of data.

An alternative is logical removal, in which the audit log is not physically changed. Instead, a logical view is computed which is consistent with the retention policy. Logical sanitization can support multiple distinct retention policies that can be associated with users or groups of users, in a manner very similar to an access control policy, which physical deletion is unable to support. Under logical log sanitization, our retention policies can be seen as a combination of fine-grained and view-based access control over a transaction-time database.

By the semantics of preservation rules, we always physically enforce them. This fact applies to all preservation rules with no exceptions. But whether to logically enforce them is a choice made by compliance officers. For example, if preservation rules are the requirements of external authorities, the compliance officer first prevents physical deletion of the related tuples to maintain the data hold. At the same time, if he thinks that these preservation rules are beyond the auditor's accessibility, he will probably choose to ignore them in the logical application, which means allowing logical deletion of the tuples matching these preservation rules. By doing this, the system

provides better privacy guarantees while still satisfying data holds. Alternatively, the compliance officer can add a data hold purely for the sake of preserving information of interest to auditors. In this case, he will enforce the preservation rule both physically and logically. But one should be warned that in this circumstance data could also be exposed to external authorities and thus data disclosure is not prevented with respect to them.

Based on the discussion above, we believe a hybrid method of history sanitization, instead of purely physical or logical application, would better accommodate common scenarios. Assume $\mathcal{P}$ is the set of rules defined on the database, containing preservation rules $\mathcal{H}$ and removal rules $\mathcal{R}$. The compliance officer can adopt a hybrid approach as follows. Physical application is always executed on all preservation rules $\mathcal{H}$ since it does not prevent logical application of these rules later. Physical application on removal rules should be carefully chosen as logical sanitization is not available for them subsequently. So the officer selects a subset consisting of important removal rules $\mathcal{R}_1 \subseteq \mathcal{R}$ for physical sanitization. Next, in the logical application step, for each group of users/auditors, he decides what information to retain for them and enforces a corresponding subset of preservation rules $\mathcal{H}_1 \in \mathcal{H}$. For removal rules, he decides what to be deleted logically for each group of users/auditors, and enforces a proper subset of non-physically-applied removal rules $\mathcal{R}_2 \subseteq \mathcal{R} - \mathcal{R}_1$.

In Section 3.8 we describe the implementation of our policies both physically (using an update program that transforms stored tables), and logically (by rewriting incoming audit queries to return answers in accordance with the stated policy).

3.5 Audit Queries under Retention Restrictions: A Tuple-independent Model

Under a retention policy that includes a redaction rule, audit queries must be evaluated over tables containing variables in place of some concrete values, i.e., this table

contains *incomplete information* or *uncertainty*. In this section we discuss the tuple-independent model (TI), using techniques for querying incomplete information [51] to describe precisely the answers to audit queries under retention policies. The major benefit of TI is that there is no additional cost of deciding certain and possible tuples because extended relational operators can compute and label each tuple explicitly on the fly.

3.5.1 Incompleteness in relations and t-relations

Both regular relations and transaction-time relations can be incomplete. There are two main features that distinguish an incomplete relation from a concrete relation. The first is the presence of variables in attribute values. The second is a *status* column, included in the schema of every incomplete relation. The status column is **C** when the tuple is *certain* to exist in the relation, and **P** when the tuple may possibly exist.

Under a retention policy $\mathcal{P}$, the inputs to our audit queries are the audit log table $\mathcal{P}(L_S)$ and t-relation $\mathcal{P}(T_S)$. Both tables may be incomplete, since they may contain variables. In addition, each of their tuples is understood to have a status of *certain*. In general, audit query answers will include both possible and certain tuples.

An incomplete relation represents a set of possible relations. Let R be a relation schema (regular or transaction-time) and let I_R be an incomplete relation over R. Also let $I_R = I_R^p \cup I_R^c$ where I_R^c are the certain tuples and I_R^p are the possible tuples. If V is the set of variables appearing in R, and f is a one-to-one function from the variables V into the domain of R, then a possible world consists of the certain tuples under f, plus any subset of possible tuples under f. Thus, the set of possible worlds represented by I_R, denoted $rep(I_R)$, is defined as:

$$rep(I_R) = \{f(I_R^c) \cup X \mid f \in F, X \subseteq f(I_R^p)\}$$

where F is the set of all one-to-one functions that assign values in the relevant domains to variables in V and $f(I_R)$ is the relation after replacing variables according to f.

Recall that in our framework, variables only appear in attributes of the client schema – not in time stamps. Extending the definition of t-relation from Section 3.3, an incomplete t-relation over S is a set of $\mathit{tuples}(S) \times \mathcal{T} \times \{\mathbf{P}, \mathbf{C}\}$. A tuple $(p_1, \ldots, p_n, t, u) \in I_S$ represents the fact that tuple $(p_1, \ldots, p_n)$ is certainly active at time instant t (if $u = \mathbf{C}$) or possibly active at time instant t (if $u = \mathbf{P}$). Incomplete t-relations can also be represented as tuples $(p_1, \ldots, p_n, \mathit{from}, \rightarrow, u)$ which means that $(p_1, \ldots, p_n)$ has status u at each instant t, for $\mathit{from} \leq t \leq \rightarrow$.

3.5.2 Extended relational algebra on incomplete relations

Next we define the extended relational algebra operators on incomplete relations. The semantics of these operators is similar to the model of relational incompleteness presented by Biskup [9], but includes extensions for transaction-time. Naturally, these operators return incomplete relations, inheriting variables from the input relations and computing the status field appropriately for output tuples. We provide definitions of selection, cross-product, concurrent cross-product, and set difference. Join and concurrent-join are derived from these, and projection, union, and the time-slice operator are defined in a standard way.

Selection

Let I_R be an incomplete relation, and E be a selection condition that is the Boolean combination of comparisons of the form $R.x = c$ (for constant c) or $R.x = R.y$. Comparisons can evaluate to $\mathbf{P}$, $\mathbf{C}$, or False. If the arguments are two different constants, or two different variables, the comparison evaluates to False. The comparison of a variable with a constant evaluates to $\mathbf{P}$. If the arguments are identical variables, or identical constants, the comparison evaluates to the *status* value for the tuple. The

Boolean combination of terms is evaluated using the rules of three-valued logic where **P** is interpreted as *Unknown*, and **C** is interpreted as True.

Tuples are included in the output of the selection operator if their status evaluates to *either* **P** or **C**. When the condition E has evaluated to **P** under the comparison of a variable with a constant, this variable binding needs to be applied to the output tuple. Formally we have:

$$\sigma_E(I_R) = \{\langle f(r.*), E(r)\rangle \mid r \in I_R, E(r) = P \vee E(r) = C\}$$

The tuples returned have all non-status attributes (denoted $r.*$) with variables replaced under mapping f, and a new status field $E(r)$.

Example 3.10. Consider the selection condition $R.a = 100 \wedge R.b = R.c$. On the input relation $\{\langle dx, dy, 9, C\rangle\}$, the selection operation will return $\{\langle 100, 9, 9, P\rangle\}$.

Cartesian product

If I_R and I_S are two incomplete relations over schema R and S, the cartesian product $I_R \times I_S$ is defined as:

$$I_R \times I_S = \{\langle r.*, s.*, status\rangle \mid r \in I_R, s \in I_S\}$$

where *status* is set to $r.status \wedge s.status$.

Concurrent cartesian product

If I_R and I_S are two incomplete t-relations over schema R and S, the concurrent cartesian product $I_R \times I_S$ is defined as:

$$\begin{aligned} I_R \times^{\diamond} I_S \;=\; & \{\langle r.*, s.*, from, to, status\rangle \mid r \in I_R, s \in I_S, \\ & [r.from, r.to] \cap [s.from, s.to] \neq \emptyset\} \end{aligned}$$

where *status* is set to $r.status \wedge s.status$, $from = max(\ r.from,\ s.from)$, $to = min(r.to,\ s.to)$.

Duplicate Elimination

Duplicates (on the non-status columns of a table) can arise as a result of projection or union, as well as selection and join (because of the substitution for variables). If a tuple is both possible and certain, it is only necessary to preserve the certain version of the tuple. In general, duplicates on the non-status columns are eliminated by preserving a single tuple with a status value equal to the disjunction of all duplicates' status values. That is, it will be **C** if at least one duplicate had status **C**.

Set Difference

If I_R and I_S are two incomplete relations, then in computing $I_R - I_S$, the tuple $\langle r.*, status\rangle$ will be removed from I_R only when there exists a tuple $\langle s.*, \mathbf{C}\rangle \in I_S$ where $r.*$ and $s.*$ shares the same value or variables on each attribute. Otherwise, write $\langle r.*, \mathbf{P}\rangle$ into result when there exists a tuple $\langle s.*, status\rangle \in I_S$ where evaluation of $r.A = s.A$ (described in operator Selection section) is **P** or **C** for all attributes A in the client schema. The rest of the tuples in I_R that do not match the two cases above will remain unchanged in the result. When I_R and I_S are t-relations, we must expand the temporal intervals into instants (according to our definition of t-relation), execute the set difference, and finally coalesce them back into intervals.

Example 3.11. Recall from Section 3.1.2 that audit query A1 returns *all employees who earned a salary of 10 at some point in time*, and can be written $\Pi_{name}(\sigma_{sal=10}(T_S))$. On the incomplete t-relation shown in Table 3.3 (for which the omitted status column is uniformly **C**) we have the intermediate result of $\sigma_{sal=10}(T_S)$:

eid	name	dept	sal	from	to	status
101	Bob	Sales	10	0	100	P
101	Bob	Sales	10	100	200	P
101	Bob	Mgmt	10	200	250	P
201	Chris	Mgmt	10	300	500	C

and the final result of $\Pi_{name}(\sigma_{sal=10}(T_S))$:

name	status
Bob	P
Chris	C

3.6 Audit Queries under Retention Restrictions: A Tuple-Correlated Model

As we have seen, in the query A4 of the motivating scenario, TI is incapable of representing answers accurately due to the failure of the tuple independence assumption. In such cases, the P and C status is no longer enough to preserve a precise result. In this section, we introduce a tuple-correlated model (TC) for the purpose of more accurate auditing. TC achieves greater accuracy by maintaining correlations among tuples explicitly. It appends additional conditions to each tuple during query processing, instead of simply using "possible" and "certain" indicators. We will define the TC model and its relational algebra operations. Also we will show the benefit in terms of the expressiveness. However, the extra conditions make checking certain and possible tuples more complicated (remember in TI there is no additional cost for that) and we will discuss that in Section 3.7. Comparison with other models is investigated in Section 3.6.4.

3.6.1 Representing incompleteness

In the TC model we associate the schema with an extra column *cond*. *cond* represents a conjunction of clauses, where each clause is a variable-variable or variable-constant comparison, e.g., $X < Y$ and $Z > 5$. Consider a database D consisting of relations over schemas $R_1, R_2, \ldots$. Each schema $R_i = \{A_{i1}, A_{i2}, \ldots, A_{ij}, cond\}$. Let $\mathbb{A} = \bigcup A_{ij}$. We define a function $h : \mathbb{A} \to T$ to classify each attribute into some data type, where T is the set of all *data type*s. In TC, all the values (variables) in the same column have the same data type, and values (variables) are only allowed to compare with those of the same type. For example, attributes *salary* and *bonus* are of the same type and are comparable.

Side conditions $\eta(D)$ (or $\eta(I_R)$) are defined for database D (or relation I_R, the incomplete relation over schema R). $\eta(D)$ (or $\eta(I_R)$) is a conjunction of inequalities which captures the distinctness among variables of the same type. The definition of the side condition conforms to the semantics of retention policies, and captures constraints that apply to all the tuples, in contrast to tuple-level conditions in the *cond* column. $v(D)$ (or $v(I_R)$) represents all the variables involved in D (or I_R). For each variable x, $dom(x)$ represents the domain of the variable. Usually when a variable is corresponding to some attribute, $dom(x)$ is the domain of that attribute. Example 3.12 illustrates a relation r in TC.

Example 3.12. $r=$

`name`	`sal`	cond
Bob	x	$x < 50$
Chris	y	true

$\eta(r) = \{x \neq y\}$, $v(r) = \{x, y\}$, $0 \leq dom(x) = dom(y) \leq 100$

An *assignment* for database D is a mapping from all variables in $v(D)$ to their domains, i.e., $\forall x \in v(D), f(x) \in dom(x)$. An assignment f for database D is *qualified*

when $f \models \eta(D)$. A set of tuples S is a *possible world* represented by D if and only if there is a qualified assignment f for D and S is equal to the set of tuples when we replace all variables with values in D, i.e., $f(D) = S$. Thus, the set of possible worlds represented by database D, denoted $rep(D)$, is defined as:

$$rep(D) = \{f(D) \mid f \models \eta(D)\}$$

Example 3.13. For the database in Example 3.12,

$$f = \{\langle x, 10\rangle, \langle y, 20\rangle\}$$

is a qualified assignment, therefore the possible world represented by f is $f(r) =$

name	sal
Bob	10
Chris	20

TC vs. TI. TI allows variables but no tuple-level local conditions. The implicit constraint on distinctness of variables in TI is written explicitly by the side condition in TC. Another difference is that in TC you can specify domains of variables, while in TI variables are always assumed to have infinite domains in order to simplify the constraints.

TC vs. c-table. In a general c-table (conditional table) [51], each tuple is associated with a condition, which is a Boolean combination of equalities. A TC table can be viewed as an extended c-table with general local inequalities and special global conditions, as the side condition plus explicitly claimed variable domains can be written as global condition in a c-table. In our application, the side conditions are usually from retention policies and tuple-level conditions are generated by queries, thus TC separates the two distinct types of constraints.

3.6.2 Extended relational algebra

We now describe a slightly different extended relational algebra compared to TI (Section 3.5.2), since we have to incorporate conditions in the query evaluation. The semantics of relational operators are defined as follows. Let I_R, J_R and I_S be tables in database D. Note that the side condition of D remains unchanged after query evaluation.

$$
\begin{aligned}
\Pi_A(I_R) &= \{\langle r.A, r.cond\rangle \mid r \in I_R\} \\
\sigma_E(I_R) &= \{\langle r.*, r.cond \wedge E(r)\rangle \mid r \in I_R\} \\
I_R \times I_S &= \{\langle r.*, s.*, r.cond \wedge s.cond\rangle \mid r \in I_R, s \in I_S\} \\
I_R \times^{\diamond} I_S &= \{\langle r.*, s.*, from, to, r.cond \wedge s.cond\rangle \mid \\
&\qquad r \in I_R, s \in I_S, [r.from, r.to] \cap [s.from, s.to] \\
&\qquad \neq \emptyset\} \text{ where } from = max(r.from, s.from), \\
&\qquad to = min(r.to, s.to) \\
I_R \cup J_R &= \{\langle t.*, t.cond\rangle \mid t \in I_R \vee t \in J_R\}
\end{aligned}
$$

In projection (Π), we always preserve the *cond* column in the result. For those duplicates with the same non-*cond* attribute values but no *cond* formula, we could combine them into a single tuple by taking the disjunction of all *cond* formulas. In TC, we choose not to do this in order to keep each formula succinct. This does not change the semantics of queries and relations. In selection (σ), we return all non-cond attribute values (denoted as $r.*$) and extend the *cond* column by a conjunction with the selecting condition E, which itself is a conjunction. If the selection condition is $E = E_1 \cup E_2$, we will execute two selection operations followed by a union operation. Cross product ($\times$) is defined by combining tuples in two inputs and taking the conjunction of their *cond* columns. Concurrent cross product ($\times^{\diamond}$) is computed in a

similar way as in TI model, plus the process on *cond* columns as in a normal cross product defined above.

3.6.3 Expressiveness

In the context of incomplete databases, expressiveness measures the ability to represent sets of possible worlds. A data model is said to be *complete* [93] when it can represent any set of possible worlds. TI is not complete because it is impossible to represent a set of possible worlds in which two possible tuples are mutually exclusive, as shown in query A4 in the motivating scenario. However, TC is complete.

Theorem 3.2. *TC is a complete data model, i.e., any set of possible worlds can be represented by a TC table.*

Proof. Assume we have any set of possible worlds $W = \{r_1, r_2, \ldots, r_n\}$. Now we construct a table R in TC: we generate a new relation by adding each tuple in every possible world, as well as distinguishing them by appending *cond* condition $z = i$ for the ith possible world. Variable z has $dom(z) \in [1, n]$. Therefore, any qualified assignment $f(z) = i$ will only associate with the ith possible world. It is obvious that the relation R represents the exactly same set of possible worlds of W. □

We say that model A is at most as expressive as model B ($A \preceq B$) if for any relation a in A there exists some relation b in B such that $rep(a) = rep(b)$ where $rep()$ denotes the set of possible worlds represented by the relation. A is as expressive as B if and only if $A \preceq B \wedge B \preceq A$. From the theorem above and the fact that TI cannot capture tuple correlations, we have the following corollary.

Corollary 3.3. *TC is more expressive than TI, i.e., TI $\preceq$ TC and TC $\not\preceq$ TI.*

3.6.4 Comparison with Other Models

Our approach combines methods from the uncertain database and temporal database communities. Some known models (Gadia [31] and Koubarakis [58]) provide a foun-

dation for this area. A recent model U-relation [1] for uncertain databases is efficient and expressive. Although it is not designed for temporal databases, we can extend the relations with time information, such as *from* and *to* columns, to answer audit queries under retention policies. Next we are going to compare TI and TC with these models and show the advantages of TI and TC.

Gadia's model [31] It is surprising that by only using nulls instead of variables in TI (Let's call it TI^{null}), the extra status column helps us gain the same expressive power as Gadia's model (proof omitted here). This fact highlights our model TI, because Gadia's model is not in the normal form of the relational database and is difficult to implement.

Example 3.14. Here is an example where Table 3.5 in Gadia's model and its equivalent representation Table 3.6 in TI^{null}.

name		dept		sal	
Bob	[0,100]	Sales	[10,30)	10	[10,40)
		Mgmt	[30,55)	12	[40,45)
	[0,50]		[0,50]		[0,50]
	[0,100]		[0,100]		[0,100]

Table 3.5: A table in Gadia's model

name	dept	sal	from	to	status
Bob	NULL	NULL	0	10	C
Bob	Sales	10	10	30	C
Bob	Mgmt	10	30	40	C
Bob	Mgmt	12	40	45	C
Bob	Mgmt	NULL	45	50	C
Bob	Mgmt	NULL	50	55	P
Bob	NULL	NULL	55	100	P

Table 3.6: The equivalent table in TI^{null}

U-relation [1] U-relation is a c-table styled data model for incomplete databases. Every relation (named U-relation) is a restricted c-table, in which the global condition

is always "true", variables are only allowed in the condition column and the local condition must be a conjunction of formula $x = a$ where x is a variable and a is a constant. Each possible world is defined by a total function $f : V \rightarrow \mathbf{N}$, where V is the set of variables appeared in the table. The variables in U-relations are used for two purposes: representing possible worlds by valuation and encoding correlation between tuples. U-relation is a complete data model, as our TC model, capable of representing any set of possible worlds.

Consider utilizing this model for audit queries under retention policies. We should enumerate all the possible assignments of each variable and list each of them as one tuple in U-relation, e.g., the tuple $(Bob, x, true)$ in TC that tells us Bob has a salary x will be split into 10 tuples if the domain size of x is 10. In the query evaluation, we also need to modify the semantics of operators in order to guarantee the distinctness property among variables. For example, when joining two tables, the query processing matches the tuples that satisfy the join condition and then takes extra effort to filter out the inconsistent combinations. The join operation in U-relation is defined as $R \bowtie_{\varphi} S = R \bowtie_{\varphi \wedge \omega} S$ where ω-condition is used to remove inconsistent tuples, i.e., one variable in two tuples is assigned with two different values. To make this join operation working in our model, we need to rewrite the ω-condition to additionally exclude the cases when two distinct variables are assigned with the same value. Let $\overline{C}$ be the set of conditions in each tuple in the form of "$\underline{var}iable = \underline{val}ue$". And $\leftrightarrow$ can be read "if and only if".

$$\omega = \bigwedge_{c_1 \in R.\overline{C}, c_2 \in S.\overline{C}} (c_1.var = c_2.var \leftrightarrow c_1.val = c_2.val)$$

The main advantage of U-relation is its simple representation and efficient implementation (incorporating state-of-art relational database systems). However, when our variables tend to have large domain size, you can not avoid listing all of those possible values by simple equations attached to each tuple in U-relation. In TC, there

are variables in the attributes, as well as variables only existed in the *cond* column. Moreover, inequalities and equalities are both allowed for *cond* formula. Such richness enables TC to represent uncertainty in a concise form. The vertical decomposition (attribute-level uncertainty) in U-relation has no conflict with the semantics of TC and it is possible to incorporate for further succinct representation.

Koubarakis' model [58] Koubarakis proposed a constraint-based incomplete temporal data model, which integrates global and local inequality constraints on the occurrence time of an event. With constraints only on temporal information, it supports indefinite instants. For example, we can say this event a happens between time 0 and occurence of event b, but we do not know its exact time. Such information related to event ordering is not allowed in Gadia's model. However, unlike Gadia's model, without variables in traditional attributes, we cannot represent the information that allows attribute values to be unknown in the (possible and certain) active period of that tuple. We basically do not allow incompleteness in temporal columns for both TI and TC, which simplifies the computation by avoiding complicated event ordering problems in the query processing. We believe that Koubarakis' model has the potential for building a more powerful system, and we save this as our future topic.

3.7 Complexity

For the TI model, as we have seen, certain and possible tuples are decided by the *status* column. However, we will show the problem of checking if a given tuple in TC is possible is NP-complete in Theorem 3.4. And Theorem 3.7 states that deciding a certain tuple is coNP-complete. However, we show in Theorem 3.5 and 3.6 that for a large subclass of instances, the possibility problem is in polynomial time. The certainty problem remains hard even within subclasses, therefore we use an exhaustive search with heuristics to compute certain tuples.

3.7.1 Deciding possible tuples

Given a TC table, there are usually tuples whose *cond* formulas are unsatisfiable, which means their existence is impossible. Computing possible tuples is the process of eliminating those unsatisfiable tuples. We begin with the definition of database satisfiability and possible tuples.

Definition 3.15. A database D in TC is *satisfiable* when it has a qualified assignment.

It is clear satisfiability of D is decided by checking satisfiability of $\eta(D) \wedge \bigwedge_{x \in v(D)} dom(x)$, where $\eta(D)$ is the side condition of the database which indicates the distinctness among variables of the same type and $\bigwedge_{x \in v(D)} dom(x)$ defines all of the variable domains in D.

Definition 3.16. A tuple t is a *possible tuple* in database D when there exists a qualified assignment f such that $f \models t.cond$.

Recall that a qualified assignment f of database D satisfies $\forall x \in v(D), f(x) \in dom(x)$ and $f \models \eta(D)$. Thus, it is easy to see that deciding possibility of the tuple t is equivalent to the satisfiability problem of the following formula:

$$\psi(t) = t.cond \wedge \eta(D) \wedge \bigwedge_{x \in v(D)} dom(x)$$

Of course the satisifiability of database D is a necessary condition for the satisfiability of any of its tuples. When we know that D is satisfiable, we can simplify the above condition $\psi(t)$ by replacing D with the current tuple t, i.e., $t.cond \wedge \eta(t) \wedge \bigwedge_{x \in v(t)} dom(x)$. Here $\eta(t) \wedge \bigwedge_{x \in v(t)} dom(x)$ are side conditions and domains only related to variables involved in $t.cond$. The simplified $\psi(t)$ is equivalent to the original one when database D is satisfiable: assignments to variables not in $t.cond$ will not change the satisfiability of tuple t. For simplicity we assume that all of the variables in $t.cond$ have the same type.

Theorem 3.4. *Given a database D in TC, and tuple $t \in D$, deciding whether t is a possible tuple is NP-complete.*

The proof is a reduction from the clique problem, which is known to be NP-hard.

We next show that two natural restrictions of the satisfiability problem can be solved in polynomial time. We avoid the richness of constraints that leads to the above NP completeness by restricting the kind of constraints that can occur at the same time, namely we do not allow constraints to simultaneously express ordering, e.g, $X < Y$, and distinctness from a given constant, e.g. $X \neq C$. Recall that distinct variables are required to take distinct values. The two subclasses of TC corresponding to these restrictions are named $\text{TC}_{<}$ and $\text{TC}_{\neq}$.

Theorem 3.5. *Given a database D in $TC_{<}$, and tuple $t \in D$, deciding whether t is a possible tuple is in P.*

Proof. We first rewrite the $\psi(t)$ as an *H-representation* of tuple t, H_t, consisting of two different sets of inequalities $H_{t,1}$ and $H_{t,2}$. 1) $H_{t,1}$: inequalities like $X < Y$. Since inequality $X < Y$ is equivalent to $X \leq Y - 1$, we only need $<$ to represent the relationship between variables ($X \neq Y$ is implicit since we have X and Y the same type). These inequalities define a topological ordering of variables. 2) $H_{t,2}$: inequalities like $X \in [X^L, X^R]$. The lower bound of X is noted as X^L while X^R is the upper bound. To compute the lower and upper bound of each variable, we take advantage of the transitive property of $<$-relationship. For example, if $X > 5 \wedge Y > X$, we have $Y > X > 5$ and because $X \neq Y$ we further have $Y^L = 7$. That is, if $X < Y$, Y^L will be updated to $max(Y^L, X^L + 1)$ and X^R is updated to $min(X^R, Y^R - 1)$. This can be done by selecting variables in a topological ordering and inverse topological ordering.

It is clear that the H-representation H_t is equivalent to $\psi(t)$. Now we are ready to create a scheduling problem such that there are n unit-time jobs (n equals the

number of variables in H_t) with release times (X^Ls in $H_{t,2}$), deadline times (X^Rs in $H_{t,2}$) and arbitrary precedence constraints (defined by $H_{t,1}$). It is easy to see finding a feasible schedule for this problem is equivalent to our tuple satisfiability in $\text{TC}_<$. Since computing H_t is in P and finding a feasible schedule for this problem is in P [59] , the tuple satisfiability can be solved in polynomial time.

□

Theorem 3.6. *Given a database D in $TC_{\neq}$, and tuple $t \in D$, deciding whether t is a possible tuple is in P.*

Proof. Consider the H-representation of $\psi(t)$, different from H_t in $\text{TC}_<$, we will have an empty $H_{t,1}$ since there is no variable comparison and each variable has a union of sets of intervals in $H_{t,2}$, instead of a single interval as in $\text{TC}_<$. For example, $X > 1 \wedge X < 10 \wedge X \neq 5$ will result in $X \in [2, 4] \cup [6, 9]$. Now we are ready to create a scheduling problem such that there are n unit-time jobs (n equals the number of variables in H_t) with multiple release and deadline times (defined by the intervals in H_2). It is easy to see that finding a feasible schedule for this problem is equivalent to our tuple satisfiability in $\text{TC}_{\neq}$. Since computing H_t is in P and finding a feasible schedule for this problem is in P [97], the tuple satisfiability can be solved in polynomial time. □

Remark 1 Recall that we simplify tuple satisfiability by assuming the database is satisfiable. Actually database satisfiability is a special case of Theorem 3.6 because the satisfiability of formula $\bigwedge_{x \in v(D)} dom(x) \wedge \eta(D)$ does not contain inequalities like $X < Y$. Thus, deciding database satisfiability can be done in polynomial time. In addition, when uncertainty is generated by applying retention policies defined in this chapter, the database is always satisfiable.

Remark 2 Consider the subclass $\text{TC}_{=\neq}$ consisting of TC restricted to conditions of the form $X \neq C$ and $X = C$ in $t.cond \wedge \bigwedge_{x \in v(t)} dom(x)$. Such conditions are common

for variables in enumerative domains in which there is no ordering among values, e.g., department type. Recall that $\text{TC}_{\neq}$ allows all kinds of variable-constant comparisons, but not variable-variable comparisons. Since $\text{TC}_{=\neq}$ is a special case of $\text{TC}_{\neq}$, we could use the algorithm for $\text{TC}_{\neq}$. Nevertheless, we can do it faster using an alternative method. Since there is no ordering among variables and constants, all variables in $t.cond$ are treated equally. We can randomly pick one variable X and assign it a qualified constant C such that C has not been assigned to other variables and $X \neq C$ does not exist. If all variables are assignable, then it is satisfiable, otherwise it is not.

Remark 3 When there are multiple variable types for tuple t, we classify inequalities by data types. As long as the constraints concerning each distinct data type fall entirely in $\text{TC}_{<}$ or $\text{TC}_{\neq}$, we can always compute possible tuples in polynomial time. For example, $t.cond \equiv X_{sal} < Y_{sal}$, $Z_{dept} \neq$ 'HR' where $X_{sal} < Y_{sal}$ is in $\text{TC}_{<}$ (the salary type) and Z_{dept} is in $\text{TC}_{\neq}$ (the department type).

Remark 4 A combination of $\text{TC}_{<}$ and $\text{TC}_{\neq}$ provides adequate expressiveness for our purposes. $\text{TC}_{<}$ is well-designed for ordered domains (e.g., integer domains like *salary*) and $\text{TC}_{\neq}$ is suitable for unordered domains (e.g., enumerative domains like *department*). If we consider the WHERE clauses of the TPC-H queries, each can be represented in $\text{TC}_{<}$ and $\text{TC}_{\neq}$ under any of our retention policies. This suggests that in many cases NP-hardness is not a practical problem. Nevertheless, when the general TC model cannot be avoided, we have to search the space for satisfiable assignments, and the complexity bound is exponential in the number of variables, n, which is bounded by a property of the schema. Variables are generated by redaction policies and n cannot exceed the number of columns belonging to the same data type, thus we expect to see n very small. In the TPC-H workload, n cannot be larger than ten. With this small number of variables, complexity exponential in n will be feasible and add very limited additional burden when compared to $\text{TC}_{<}$ and $\text{TC}_{\neq}$.

3.7.2 Deciding certain tuples

A tuple is certain when it occurs in every possible world represented by the database.

Definition 3.17. Suppose we are given a database D, and the set of possible worlds represented by D, $W = \{f_1(D), f_2(D), \ldots\}$ where $F = \{f_1, f_2, \ldots\}$ is the set of all qualified assignments for D. A tuple t is *certain* iff it exists in every possible world $f_i(D)$, for any $f_i \in F$.

As each possible world contains only constants, we can infer that a certain tuple contains no variables. In addition, if a tuple exists in every possible world, its *cond* formula should be always satisfiable for all qualified assignments. To compute the certain tuples in a TC table, we have a two-step process. First, we compute the *certain v-tuples*. A certain v-tuple is a relaxed version of a certain tuple, meaning its *cond* formula is always satisfiable but it could have variables in some columns. We merge tuples with the same non-*cond* column into a new tuple t and generate the new $t.cond$ formula by making a disjunction of all the *cond* formulas. Then if $f(t.cond) = true$ for every qualified assignment f, t is a certain v-tuple. Second, we transform certain v-tuples to certain tuples. Obviously, a certain v-tuple is a certain tuple when it does not contain variables. When t has a variable on attribute A, it can be transformed to a certain tuple if and only if there are another $|dom(A)| - 1$ certain v-tuples with different variables of A. In other words, there are at least $|dom(A)|$ certain v-tuples with distinct variables of A, therefore, by distinctness of variables of the same data type, each corresponds to a certain tuple. When the size of database is unbounded, the difficulty of the first step dominates, since step two can be computed efficiently.

Theorem 3.7. *For each variant of the TC model discussed above, namely TC, $TC_<$, $TC_\neq$ and $TC_{=\neq}$, the tuple certainty problem is coNP-complete.*

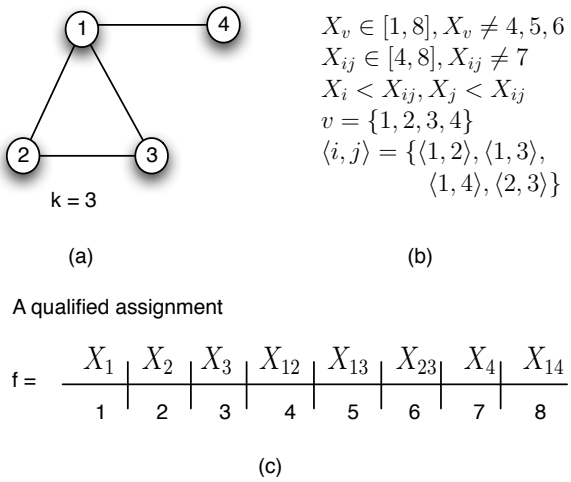


Figure 3.2: Reduction from clique problem to tuple satisfiability problem in TC_{all}.

The proof involves reductions from the 3DNF tautology problem to prove the coNP-completeness for the TC classes.

We can do an exhaustive search to detect certain tuples by a backtracking method. We choose a variable, assign a value, and then simplify the formula, recursively checking if it is still a certain tuple. Assume a variable's *valid intervals* consists of the union of all intervals in which the variable could find a qualified assignment that makes at least one of the conditions in *cond* true. One necessary condition for a certain tuple is that each variable should have its valid intervals equal to its domain. Thus, in each recursive step, if any variable has a smaller valid interval than the domain, the tuple is not a certain tuple. This heuristic will help to eliminate non-certain tuples quickly.

The worst case complexity of calculating certain tuples is primarily determined by a term exponential in the number of variables n. Similar to our discussion in Remark 4 about possible tuples, n tends to be a small number. Thus, in practical cases, e.g. for schemas like TPC-H, the overhead added here is limited.

3.7.3 Proof of the Theorems of TC

Here we finish the proofs of the theorems for TC in Section 3.7.

Proof of Theorem 3.4

Proof. The problem is in NP because given an assignment, we can verify in polynomial time whether t's formula is evaluated to True. Now we are going to generate a reduction from the clique problem to show it is NP-complete. Consider an instance of clique problem: a graph $G = (V, E)$ and $k = 3$. Let $V = \{1, \ldots, n\}$ and $m = |E|$, we build a tuple t that consists of $n + m$ variables: $X_i : i \in V$; $X_{ij} : (i, j) \in E$. These variables must take distinct values in the range $[1, n + m]$. Furthermore we will write inequalities so that the first k values are taken by vertex variables X_is and the next $\binom{k}{2}$ values are taken by edge variables, and the next $n - k$ values are taken by the rest of vertex variables. Furthermore, we will write inequalities requiring that for each edge (i, j), $X_i < X_{ij}$. Thus the conditions of the inequalities will be satisfiable iff G has a k clique. Formally we define $t.cond$ as follows,

- For each vertex i in V, we create a *vertex* variable X_i s.t. $X_i \in [1, n + m]$ and $X_i \neq k + a$ for all $0 < a \leq k(k - 1)/2$
- For each edge (i, j) in E, we create a *edge* variable X_{ij} s.t. $X_i < X_{ij}$, $X_j < X_{ij}$, $X_{ij} \in [k + 1, n + m]$, and $X_{ij} \neq k + k(k - 1)/2 + b$ for all $0 < b \leq n - k$.

An example of the reduction on one small example is shown in Figure 3.2. It is clear this reduction is polynomially bounded. To prove the correctness of the reduction, we will see the tuple is satisfiable if and only if G has a k-sized clique. When there is a k-sized clique in G, we can always find a qualified assignment that makes the tuple satisfiable. From 1 to $n + m$, we put the variables into slots by the following order: k vertex variables in the clique, $\binom{k}{2}$ edge variables in the clique, $n - k$ vertex variables and rest of edge variables. Consider the case k-sized clique does not exist in G. By construction, slots from $k + 1$ to $k + k(k - 1)/2$ are only allowed for edge variables. Since there are less than $\binom{k}{2}$ edges connected among any set of k vertices, at least one of these slots is empty. With domain size equal to the number

of variables, we can conclude the tuple is not satisfiable. Therefore the reduction is correct and the theorem holds.

□

Proof of Theorem 3.7

Proof. All the problems are in coNP because given an assignment in polynomial time you can verify the tuple is not a certain tuple when t's formula is evaluated to False. To prove coNP-completeness, we construct a reduction from 3TAUT – the 3DNF tautology problem, which is known to be coNP-complete. For any 3DNF instance A with n variables, we will construct a table T in TC. The relation is a set of single-column tuples. Each tuple has a value 100 and the number of tuples is equal to the number of clauses in A. We create n variables with the same type for relation T and a mapping f from variables in A to variables in T. Each variable in T has an index associated with it, i.e. from x_1 to x_n. Variables in relation T have domain $[1, n+1]$. Each tuple corresponds to a clause, and add an inequality $x_i \leq i$ ($x_i > i$) in the *cond* column for each positive literal (negative literal). Consider

$$A \equiv (\bar{x} \wedge y \wedge z) \vee (x \wedge \bar{y} \wedge \bar{w})$$

we construct the T with variables x_1, x_2, x_3 and x_4.

value	**cond**
100	$x_1 > 1, x_2 \leq 2, x_3 \leq 3$
100	$x_1 \leq 1, x_2 > 2, x_4 > 4$
	$\eta(D) \equiv x_1 \neq x_2 \neq x_3 \neq x_4$
	$dom(x_i) = [1, 5]$

It is clear the construction is polynomially bounded. Now we prove the correctness of reduction. 1) If A is not a tautology, there must exist an assignment to the

variables in A such that none of clause can be evaluated True. Assume the assignment is $\{a_1, \ldots, a_n\}$ where $a_i = 0$ or 1. We build a qualified assignment $\{b_1, \ldots, b_n\}$ to variables in T, where $b_i = i$ if $a_i = 1$ and $b_i = i + 1$ if $a_i = 0$. $\{b_1, \ldots, b_n\}$ will not make any tuple's condition true, otherwise the corresponding clause in A will also become true under assignment $\{a_1, \ldots, a_n\}$, which is a contradiction. 2) If tuple $\langle 100 \rangle$ is not a certain tuple, i.e., there must exist some qualified assignment to variables in T such that none of tuple's conditions is true under this assignment. Assume the assignment is $\{b_1, \ldots, b_n\}$ where $b_i \in [1, n + 1]$. We build an assignment $\{a_1, \ldots, a_n\}$ to variables in A, where $a_i = 1$ if $b_i \leq i$ and $a_i = 0$ if $b_i > i$. Similarly, $\{a_1, \ldots, a_n\}$ will not be valid to any clause, otherwise the corresponding tuple's condition will be true by assigning $\{b_1, \ldots, b_n\}$. Therefore, the reduction is correct and polynomially bounded. The T relation created here only has variable-constant comparisons and no $\neq$ operator in *cond* column, so it is an instance of TC, $\text{TC}_{<}$ and $\text{TC}_{\neq}$. So coNP-hardness holds for all of three different classifications. In addition, for $\text{TC}_{=\neq}$ we could have another reduction from 3DNF tautology problem. Consider the same formula A as above, we construct a table T in $\text{TC}_{=\neq}$ and the correctness of reduction can be proved similarly.

value	**cond**
100	$x_1 \neq 1, x_2 = 2, x_3 = 3$
100	$x_1 = 1, x_2 \neq 2, x_4 \neq 4$
	$\eta(D) \equiv x_1 \neq x_2 \neq x_3 \neq x_4$
	$dom(x_i) = [1, 4]$

Moreover, since T in our construction only has one type of variable, deciding certain tuples with variables of multiple types is also coNP-complete.

□

3.8 Implementation

The implementation of our framework translates our historical data model into standard relations in Postgres. Our goal is to show the practical feasibility of our framework. We optimize our implementation using commonly-available indexing strategies and query rewriting techniques. A fully optimized implementation might make use of techniques specifically designed for transaction-time data, but these are beyond the scope of our prototype. Note the earlier implementation described in [67, 68] only includes the TI model.

As a performance optimization, both the audit log and the transaction-time tables are stored in our implementation. As noted earlier, the transaction-time tables are redundant since they can be computed from the audit log. However, materializing these tables and maintaining them upon changes to the log eases query expression and evaluation for some audit queries. The efficiency gains seem well worth the space overhead which is roughly double that of storing the audit log alone. The time stamp fields $from$ and *to* are combined into one attribute named *trange*, which is stored as an interval type (actually a one-dimensional cube data type in Postgres). Utilizing the cube data type simplifies the expression of the concurrent join, and we also use an available R-tree implementation. In TI, *status* is represented as a Boolean value. In TC, we split the conjunction in the *cond* formula and put each inequality (or equality) into a text value column.

Recall that the application of policies can be executed either physically or logically (see the discussion in Section 3.4.5). In the remainder of the section we discuss the physical application of retention policies followed by query evaluation on physically sanitized datasets. Then we describe the logical application of policies. Lastly we discuss the computation of possible and certain tuples.

3.8.1 The physical application of retention policies

The application of retention policies is implemented by transforming the input rules into a set of update operations on the original t-relation and possibly the audit log. Inconsistencies may arise if the subsequent application of new policy rules touches the previously sanitized attributes [4, 98]. For example, one policy p_1 removes the department information and the other policy p_2 hides employees' salary in the HR department. Applying p_1 first will result in a different sanitized history than if p_1 is applied second. In this example, p_2 will remove nothing if p_1 is already applied, however, the sanitized history will be different if p_1 is applied first. To avoid this, we assume we have all the policy rules at the time of policy application. Policy application for all rules is accomplished in one-pass scanning of the table, sanitizing each tuple against all rules, which guarantees that all the conditions in the rules are fully evaluated on the current tuple before removing any values from that tuple. For example, if an employee is in the HR department, both his salary and department information will be deleted when we have p_1 and p_2.

Redaction is implemented by replacing values with variables. As described previously, variables here preserve equality even after redaction. That is to say, the relationship between value and variable is a strict one-to-one mapping. In our current implementation, we use a cryptographic hash function. Specifically, each data type has a distinct hash function, which allows consistent variable assignments on the same values even across multiple tables. Remember that we define the data type when introducing TC in Section 3.6 and this concept can also be applied on TI. As a data type only relates to the comparability and does nothing with the domain, another benefit of utilizing a hash function for each data type is to enable comparison of variables that belong to different attributes, e.g., comparing *salary* and *bonus*.

Since the policies are specified over t-relations, a policy $\mathcal{P}$ with an arbitrary time condition $[u, v]$ may require a split of update intervals causing phantom updates in

the sanitized log (as demonstrated from Example 3.7 and Table 3.4), which results in residual disclosure and false conclusions in query evaluation (meaning audit answers will no longer be sound). To avoid this, we adjust the redaction period to the nearest modification period of any field. However, this method might be too restrictive and hinders periodic policy application, especially when the nearest modification period is much longer than that required by the retention policies. To favor practicality and periodic application but still achieve no residual disclosure and soundness, one possible approach is to impose a system-wide "soft" limitation of the active period of time for each tuple. As a tuple's active period is defined as $to - from$, the requirement ensures that no tuple is going to stay in the current snapshot of the database longer than the limit *roughly*. For example, if Bob's salary remains unchanged for about one year, which reaches the limit of a tuple's active life time, the system will input a new tuple with the same salary starting from some randomly selected date and archive the old tuple in the history. In the log table, all audit fields of the new tuple are copied from the last update. Thus, we can align the time with finer granularity and apply policies periodically. This randomly cyclic strategy preserves the soundness of query answering as long as we treat the system's behaviors as true updates of the history. Moreover, by assigning consistent variables on the same values, auditors can still correctly monitor the changes to values.

3.8.2 Audit query evaluation

Next we implement in SQL the semantics of extended relational operators over incomplete relations for both TI and TC. The basic strategy is to rewrite SELECT-FROM-WHERE blocks to accommodate incompleteness.

The TI model. To get uncertain answers for any given user query, the query evaluator runs over the rewritten version of that query. During query processing, it

retains tuples whose *status* column evaluates to either **P** or **C** on the original WHERE clause, and eliminates all others. Then it computes the correct *trange* (if necessary), the status column, and appropriate values of variable bindings for the query results.

In the following algorithm, the function $isvari(x)$ tests if x is a variable. $onevari(x, y)$ returns true only when one of x and y is a variable. $binds(x, y)$ represents the value bindings, described in Section 3.5. It outputs x if x is a constant, otherwise it outputs y. In addition, to simplify the representation, we assume that the WHERE clause of the user query is always a conjunction of multiple condition expressions. If there are attributes appearing in two conditions connected by the OR operator, e.g., `sal=10 OR sal=20`, we can break the query into parts and later combine their results. The algorithm for rewriting user queries is as follows:

1. WHERE clause: rewrite each condition by the following rules. $T.A$ stands for attribute A in table T. $\theta \in \{=, \neq, <, \leq, >, \geq\}$. c, c_1 and c_2 are constants.

$$A\,\theta\,c \;\Rightarrow\; (A\,\theta\,c \text{ OR } isvari(A)) \tag{3.1}$$

$$\text{Let } Z \;\equiv\; (A\,\theta\,B \text{ OR } onevari(A, B))$$

$$A\,\theta\,B$$

$$\Rightarrow\; Z \qquad (\text{if } \theta \in \{=, \leq, \geq\}) \tag{3.2}$$

$$\Rightarrow\; Z \text{ AND } A\,!=\,B \quad (\text{if } \theta \in \{<, >\}) \tag{3.3}$$

$$\text{if exists } T_1.A \;=\; c_1 \text{ and } T_2.A \,=\, c_2\ (c_1 \neq c_2)$$

$$\Rightarrow\; \text{append } T_1.A\,!=\,T_2.A \tag{3.4}$$

The general idea of rewriting a condition is to allow the query processing to keep not only those tuples satisfying the condition but also those that could *possibly* satisfy the condition when variables are involved. Rule (1) tells the query evaluator that when A is a variable it will also retain the tuple. When

comparing two attributes A and B, by rule (2), the answer is yes when $A\theta B$ is true, or one of them is a variable. If both of them are variables and the comparison is $<$ or $>$, we additionally make sure they are two different variables, by rule (3). Similarly, in rule (4), the same attribute in different tables is compared with different concrete values. Finally we also add conditions on *trange* when necessary.

2. SELECT clause: for each column A in the original SELECT clause, we rewrite it by the following rules. Assume W is the original WHERE clause.

$$\begin{aligned}
\textbf{If} \quad & A \quad \text{is } status: && (3.5)\\
\Rightarrow \quad & (W \ \mathsf{AND}\ T.status)\ \mathsf{AS}\ status\\
\textbf{Elseif} \quad & A \quad \in W \text{ and exists } T.A = c: \\
\Rightarrow \quad & c\ \mathsf{AS}\ A && (3.6)\\
\textbf{Elseif} \quad & A \quad \in W \text{ and exists } T_1.A = T_2.A: \\
\Rightarrow \quad & binds(T_1.A, T_2.A)\ \mathsf{AS}\ A && (3.7)\\
\textbf{Else} \quad & \Rightarrow \quad A && (3.8)
\end{aligned}$$

To calculate the *status* column, as shown by rule (5) above, put the original WHERE clause into SELECT clause, and add a conjunction of related status columns to the term. Rule (6) ensures the concrete value is returned if there is an equality condition on that column. We must rewrite those columns when they appear in both the SELECT list and some equality expression in the WHERE clause, in order to make sure query evaluation returns the concrete value as shown in rule (6), or the correct variable bindings for the selection as shown in rule (7). Finally, compute the correct *trange* value if necessary (i.e., for concurrent join).

Example 3.18. The following is an example query on complete table `emp`:

```
SELECT name, t1.dept, t2.sal
FROM emp AS t1, emp AS t2
WHERE t1.dept=t2.dept AND
      t1.sal=100 AND t2.sal=200
```

The algorithm above will produce the following rewritten query if `emp` is incomplete:

```
SELECT name, binds(t1.dept,t2.dept) AS t1.dept,
       200 AS t2.sal, (t1.dept=t2.dept AND
       t1.sal=100 AND t2.sal=200 AND
       t1.status AND t2.status) AS status
FORM emp t1, emp t2
WHERE (t1.dept=t2.dept
      OR onevari(t1.dept, t2.dept))
      AND (t1.sal=100 OR isvari(t1.sal))
      AND (t2.sal=200 OR isvari(t2.sal))
      AND t1.sal!=t2.sal
```

We first apply rule (1) and (4) to generate the AND-terms in the new query since there are `t1.sal = 100` and `t2.sal = 200`. Rule (2) is also applied on `t1.dept = t2.dept`. Rule (8) keeps `name` in the selection list. We have `200 AS t2.sal` and `binds... AS t1.dept` by rule (6) and (7). Finally, we use rule (5) to calculate *status* column in the selection list.

As discussed in Section 3.5, duplicates may arise in the result of operations such as union, projection and join. The duplicate elimination process can be achieved by grouping on all non-status columns and then aggregating the (boolean) status column using bitwise OR.

The TC model. We apply a similar rewriting process as we used in TI. The difference is how to generate *cond* formulas in the result and eliminate unsatisfiable tuples.

Example 3.19. Given a query asking for employees whose bonus is more than his salary, we rewrite it as follows:

```
SELECT name,
     CASE WHEN isvari(sal) or isvari(bonus)
          THEN '[money]'|| sal ||'<'|| bonus
```

```
          ELSE NULL
     END AS cond_1
FROM emp
WHERE (salary<bonus OR onevari(sal, bonus))
     AND salary != bonus AND
     check_sat(cond, array[
          '[money]'|| sal ||'<'|| bonus])
```

In the SELECT clause we insert a case statement to output *cond* formulas with each condition recorded in a single column. In the example, as shown in the SELECT clause above, we refer to it as `cond_1`. Because the query is comparing *salary* with *bonus*, the condition only exists in the result when at least one of them is a variable. For example, if salary is z and bonus is 10, by the CASE statement, the produced inequality will be `[money] z<10`, because salary and bonus both belong to the data type "money". The number of inequalities generated, which is the number of CASE statements needed, is determined by the length of the original WHERE clause. (Actually we could reduce the size of the original WHERE clause by the process of computing the H-representation described in the proof of Theorem 3.5 and 3.6 when we treat each attribute name in the WHERE clause as a variable).

In the WHERE clause, as the last step before results are passed to the SELECT clause, a customized function *check_sat* is called to check tuple satisfiability by inputing two parameters: the current *cond* formula and inequalities formed by the condition in the original WHERE clause.

3.8.3 Logical policy implementation

The implementation above is based on the physical removal of expired information. We can also implement policies logically, or virtually, without altering the stored contents of the database. A query Q is not evaluated on the underlying database directly, but is first composed with the policy $\mathcal{P}$ to generate a rewritten query $Q_{\mathcal{P}}$. The rewritten query can be evaluated safely on the base relations and produces a result equivalent to evaluating $\mathcal{P}$ on a physically altered database.

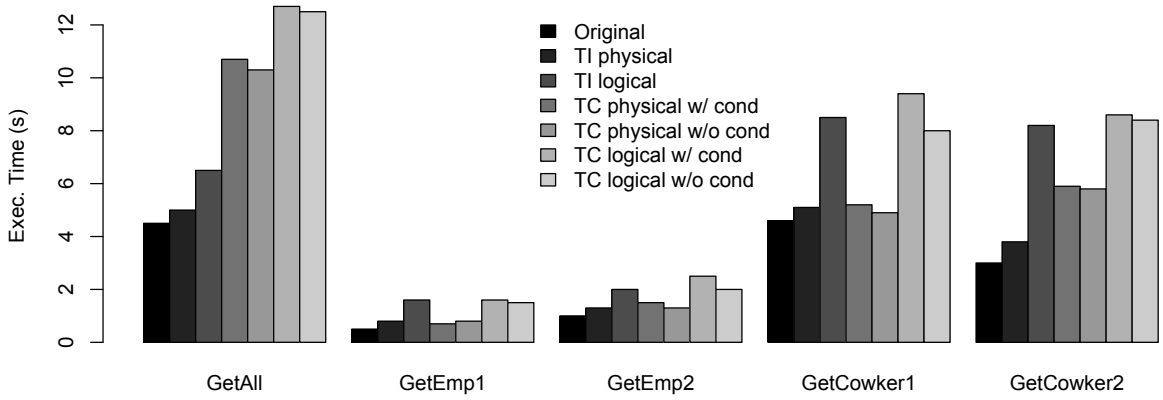


Figure 3.3: Performance of the five queries. For each query, the bars from left to right represent the execution time on different models. "physical"/"logical" means policy application is implemented physically or logically . "w/" or "w/o cond" decides whether the result will contain condition formulas.

For simplicity, we assume that the redaction policies satisfy the condition in Proposition 3.1 of Section 3.4.3, so that their application is sound and secret. Generally the composition will begin by adopting the rewriting algorithm in the previous subsection. Attributes appearing in either the SELECT or WHERE clause are called *critical attributes*. A redaction rule is *relevant* to Q when its redaction attribute list shares some attribute with Q's critical attributes. In addition to the rewriting process in the previous subsection, we also make the following changes:

1. FROM clause: for each table, add a case statement modification based on its relevant redaction rules.

2. WHERE clause: for any expunction rules $(\phi, [u, v])$, add conjunction of *not* ($\phi \wedge$ *trange* overlap $[u, v]$).

Note that the case statement modification is inspired by similar work in [60], but we change the semantics from NULLs to variables.

3.8.4 Improving query evaluation in TC

In Section 3.7, we discussed how to decide possible and certain tuples given a TC table. Consider an incomplete history D generated by the application of retention policies, and suppose we wish to compute possible and certain tuples of query q over D. Since D has no *cond* column, the conditions in the results will only be introduced by q's WHERE clause. In fact, the size of the WHERE clause might be reduced by computing its H-representation, described in the proof of Theorem 3.5 and 3.6. Each H-representation H_t contains two sets of inequalities: $H_{t,1}$, inequalities like $X < Y$ and $H_{t,2}$, inequalities like $X \in [X^L, X^R]$. The size of $H_{t,1}$ is bounded by $O(n^2)$ and the size of $H_{t,2}$ is bounded by $O(n + c)$ where n is the number of column names in q and c is the number of $\neq$-inequalities. So if there are few or no $\neq$-inequalities in q, the size of WHERE clause (the number of inequalities introduced by q) is usually very limited when the number of columns in the database D is small.

In TC, possible tuples can be checked during query processing and certain tuples are computed as a separate step after query evaluation is finished. To improve the performance, we may be able to take advantage of static analysis on the original query before rewriting and executing it on the database system. A simple example is that if a query q has only $=$-comparisons, its possible tuples are the same under TI and TC. Moreover, if this q also contains only columns not touched by retention rules, which means returned results are always complete, TI and TC result in the same set of certain tuples.

It is also possible to predetermine the satisfiability of results for some queries. Consider the WHERE clause as a formula. We first replace each attribute name with a different variable of that data type. If there exists equality between two variables $x = y$, we replace all occurrences of y with x (this is important because of the distinctness among variables of the same type). Now it is obvious the result is an empty set (no possible tuples) if this formula is not satisfiable. When this formula is

a tautology (for any qualified assignment), and all the columns in the WHERE clause are removed together by redaction policies, then all possible tuples in the result are certain v-tuples. (Recall that certain v-tuples are tuples with variables and an empty *cond* column, defined in Section 3.7.2.)

3.9 Evaluation

In this section we study the performance of query processing in our framework and evaluate the impact of retention policies on the accuracy of query results. Our experiments address the following key questions:

Performance. We assess the performance overhead of evaluating audit queries using both physical and logical policy application on TI and TC.

Accuracy of uncertain answers. We study the impact of retention policies on the accuracy of query results under TI and TC. Over sample data, we measure the precision and recall of query answers as a function of the selectivity of redaction policies. We characterize the cases where accurate auditing can be achieved under retention restrictions. And we show that TC can improve the accuracy significantly over TI in some cases. We also compare the accuracy with suppression only using NULLs. Using NULLs is a common solution in relational database research such as fine-grained access control [60]. However, variables can hide values while preserving more information about changes. We show that the extra information kept by variables significantly increases the accuracy of audit query answers.

3.9.1 Experimental setup

In all our experiments we use Postgres 8.3 running on an Intel Core2 machine with 2.26GHz CPU and 4Gb memory. Our datasets are synthetically-generated histories based on our example client schema S(eid,name,dept,sal,bonus). Here *sal* and *bonus* have the same data type which allows comparing between them.

We generated our history with an initial set of employees that grows slowly over time through periodic insertions. We apply a random sequence of independent updates to attributes throughout the lifetime of individuals. Thus the total tuples in the t-relation and log is closely approximated by the product of two parameters: the initial number of employees (the *original snapshot size*) and the average number of versions of each employee tuple (the *history length*). We measure the query execution time by reporting the average of 10 runs with the largest and smallest runs omitted.

3.9.2 Performance

We use three redaction policies[3] and five queries in our experiments. They are:

R1: (HideSal) Redact salary values for a set of departments d_{s1} before a specified time t_1.

R2: (HideBonus) Redact bonus values for a set of departments d_{s2} before a specified time t_2.

R3: (HideDept) Redact department values in a specified time period p_1.

Q1: (GetAll) Return the whole emp table.

Q2: (GetEmp1) Return employees who are in department d_1 and have salary m_1.

Q3: (GetEmp2) Return employees' information where salary is less than m_2 and bonus is larger than m_3.

Q4: (GetCowker1) Return all employees who worked in the same department as a specific employee e at the same time.

[3]We do not consider expunction and preservation rules since they will simply remove or preserve tuples and change the size of the history.

Q5: (GetCowker2) Return all employees who earned more bonus than their salary and worked in the same department with a given employee e, at same period of time, as long as the returned employees have a smaller salary than e, but a larger bonus than e.

We measure the execution time of each query under *physical* and *logical* implementation of TI and TC models. For the TC model, we also consider the case of returning results with and without *cond* formulas. The baseline (*original*) is the time to compute the audit query without the retention policy, that is, on the original tables. In GetEmp1, d_1 is in the set d_{s1}, and thus there is uncertainty in the answers due to HideSal and HideDept. Note that GetAll has no WHERE clause and GetEmp1 and GetCowker1 only contain =-comparison, therefore they can be answered accurately by TI model and thus a satisfiability check by function call in the database system is not necessary in TC. In GetEmp2, we set $m_2 < m_3$, e.g., $m_2 = 10$ and $m_3 = 40$. Consider an employee has the same salary and bonus, both redacted to variable x. Then he will not be a qualified result for GetEmp2, because $x < 10 \wedge x > 40$ is unsatisfiable. GetCowker2 has a more sophisticated situation. So rewriting GetEmp2 and GetCowker2 in the TI model can not produce results accurately. Only the TC model is able to answer these two queries properly. The execution time on a history (roughly one million tuples) with 10000 initial employees (snapshot size) and 100 versions for each one (history length) is illustrated by Figure 3.3. Generally, we find that evaluating queries under retention restrictions has a modest overhead, to be expected from the added clauses in the queries and the fact that result sizes are increased because of uncertain tuples.

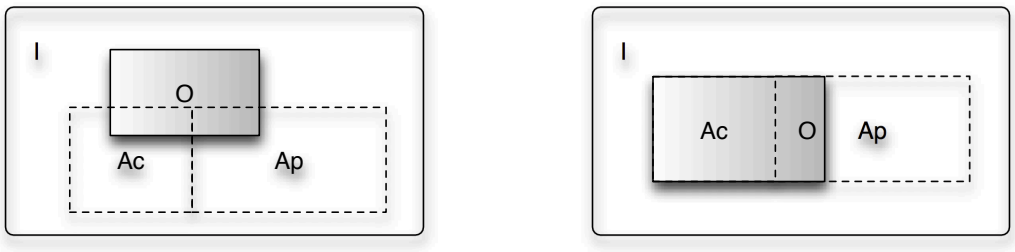
In the TC model, the online satisfiability check is implemented as a function in the plpython language in PostgreSQL. To estimate the overhead of the function call in PostgreSQL, we create a fake satisfiability check function in plpython which does nothing but returns a True value and insert it into GetAll's WHERE clause in TC. As

GetAll returns all one million tuples, the system will execute the fake function on each of them, which results in a lot of the extra cost for TC model. Considering this cost is introduced by the system and the size of result, we consider this overhead acceptable. It would be possible to reduce this overhead by using more efficient native language of the database system.

Since GetEmp1 and GetCowker1 only contain equality comparisons, checking tuple satisfiability is not needed and the only difference between TI and TC is the way they generate status and *cond* columns. As expected, the performance is very close between these two models for both physical and logical implementations. GetEmp2 and GetCowker2 add an extra cost of checking tuple satisfiability. Each tuple of GetEmp2's result has a condition consisting of at most two comparisons. In GetCowker2, there are at most four variables and the size of the WHERE clause is about eight. For these two queries, Figure 3.3 shows that computing possible (satisfiable) tuples in TC adds a modest extra cost to TI when we take into consideration the cost of the system call discussed above. When TC does not include the *cond* column in the result, the performance is closer to TI.

In addition, the logical solution is uniformly slower than the physical because of the more complex queries required when policies are composed with queries. Another reason is the lack of indexes. When a query is logically rewritten, the only usable index is the one built on *ttime* column. A possible optimization is only integrating relevant policies into query rewriting, e.g., in GetCowker1, redaction rules for removing two salary columns can be omitted from logical queries.

It is worth noting that the certain tuples alone can be computed more quickly than the original result in TI [68]. This is because, given the rewritten query, computing certain tuples can ignore variables and the certain tuple set returned tends to be smaller than the true result. In TC, computing certain tuples is a separate step after



(a) General case

(b) Guarantee of precision = 1

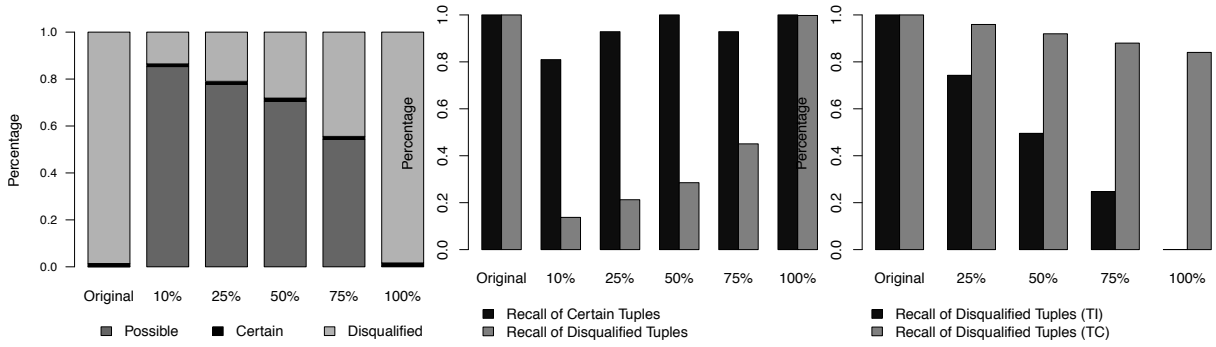
Figure 3.4: Result relationship in Venn Diagram: The answer space is I (the largest box) and the original answer are O (shaded box), the certain tuples in our model are A_c, the possible tuples is A_p (both are boxes with dotted-line).

query evaluation is done and the execution time could be slow, depending on the complexity of formulas and the number of duplicates.

3.9.3 Accuracy of uncertain answers

Next we evaluate experimentally the accuracy of audit query answers under retention policies. We demonstrate the cases that TC and TI are at the same level of accuracy and the cases when TC improves upon TI. Over the original data, an audit query can be considered to partition the set of all feasible query answers (determined by the active domain) into qualified tuples and disqualified tuples. Under retention restrictions, an audit query partitions the set of feasible answers into certain tuples, possible tuples, and disqualified tuples.

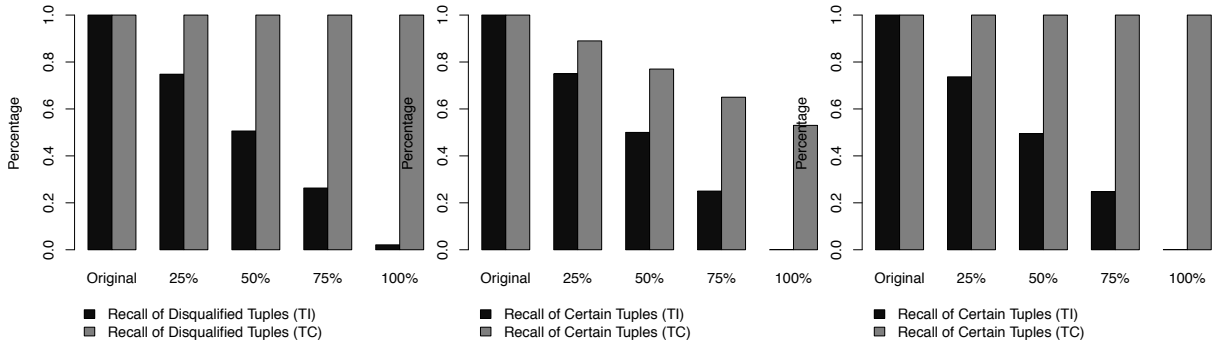
Our first measurement of accuracy considers the distribution of answers as a function of the selectivity of the redaction policies. The second measurement is the *precision* and *recall* of our answers with respect to the original answers. Assume the answer space is I and the original answer is O. The certain tuples in our model are A_c, the possible tuples are A_p. For simplicity we assume no variables in the possible tuples. Intuitively, we want to know how large $O \cap A_c$ (Fig 3.4(a)) is in proportion to



(a) Answer Distribution (GetCowker1)

(b) Recall of Answers (GetCowker1)

(c) Recall of disqualified tuples, given q = select * from emp where salary>bonus and salary<10.



(d) Recall of disqualified tuples, given q = select * from emp where salary>bonus and salary<10 and bonus>40.

(e) Recall of certain tuples, given q =select eid, dept from emp where salary<bonus .

(f) Recall of certain tuples, given q =select eid, dept from emp where salary=bonus and (salary<10 or bonus≥10) .

Figure 3.5: Accuracy of uncertain answers. We measure the accuracy (Y-axis) in terms of the removal rate of values in the history (X-axis) defined by redaction rules. In (a) and (b), we use the redaction rules defined in the previous section. (c) is performed under a rule that removes only *salary*. (d), (e) and (f) have the same redaction rule that deletes *salary* and *bonus* together.

O and A_c. Formally, the precision of certain tuples is defined by $\frac{O \cap A_c}{A_c}$ and the recall of certain tuples is defined by $\frac{O \cap A_c}{O}$.

We can also define precision and recall of the disqualified tuples, which may be relevant to auditors since they might have value in an investigation. Then $I - O$ contains the disqualified tuples in the original answers and $I - A_c - A_p$ is the set of disqualified tuples computed in the incomplete history. The precision of disqualified tuples is

defined $\frac{(I-A_c-A_p)\cap(I-O)}{I-O}$ and recall of disqualified tuples is defined $\frac{(I-A_c-A_p)\cap(I-O)}{I-A_c-A_p}$. If we consider sound and secret retention policies, as described in Section 3.4, then the precision of certain and disqualified tuples is always equal to 1, shown in Fig 3.4(b), because the soundness (Proposition 3.1) guarantees $A_c \subseteq O$ and $O \subseteq A_c \cup A_p$.

The first experiment is performed on GetCowker1 in the previous section (the concurrent self-join). The query answers in TI and TC will have exactly the same set of possible and certain tuples, although they will differ in the way they represent the condition. The answer distribution and recall are shown in Figure 3.5(a) and 3.5(b). At the beginning, there are no possible answers against the original history, and thus the recall of the certain and disqualified tuples is 1. When there are values removed by retention rules, possible answers are introduced. The percentage of possible tuples and the recall of the certain and disqualified tuples all have an inflection point as the selectivity goes up. This is because, when the removal rate is low, fewer variables are introduced so we can retain a high recall. When the rate increases, the number of variables increases and the recall decreases. On the other hand, when the rate is extremely high, the incomplete history is mostly replaced with variables on the join attribute: department. We will get high recall since the equivalence among variables can be inferred accurately, e.g., two employees both working in HR department result in the same variable x in their department attribute. Therefore, there are fewer possible answers and we get very high accuracy when all the department information is removed, similar to the answers under the original history.

There are also many queries where TC can obtain much greater accuracy. Figure 3.5(c) and 3.5(d) show the difference in recall of disqualified tuples for TI and TC given two queries. (We omit the answer distribution figures here.) Since possible tuples in the uncertain results are always coming from the disqualified tuples in original results, these two figures actually illustrate the difference in size of the possible (satisfiable) tuples. That is, TI will output more tuples which should be unsatisfi-

able and eliminated. In Figure 3.5(c), we remove history by deleting salary values. When salary is replaced with a variable X, the *cond* condition in the result will be $bonus < X < 10$, i.e., the tuple is possible only when bonus is less than 10. In TI, this fact cannot be captured. Therefore, when the number of removed *salary* attributes increases, the size of possible tuples grows and finally all the tuples are possible when the removal rate reaches 100%. However, in the case of TC, the size increases linearly because salary and bonus are generated randomly and the probability of bonus less than 10 is independent of the removal of salary. In Figure 3.5(d), the WHERE clause in the query is an unsatisfiable formula. Thus, no matter how we redact salary or bonus individually or jointly, TC will always return an empty result (as does the query over the original history) while TI increases quickly when we remove more. Note that when all of the salary and bonus attributes are redacted, TI does not return all the tuples because it eliminates the tuple where salary and bonus are replaced with the same variable. In fact, we can actually use the static analysis discussed in Section 3.8.4 for TI to avoid this but we can do nothing for the case of Figure 3.5(c).

Figure 3.5(e) and 3.5(f) show the recall of certain tuples for two different queries. In Figure 3.5(e), with condition $salary < bonus$, a certain tuple in TC has conditions such as $X < Y \vee Y < X$, or $X < 10 \vee 8 < X$ after merging tuples with identical non-*cond* columns. Thus, we can expect a smooth reduction when the removal rate increases. The result shows ten thousand certain tuples in the original history and half of the tuples in the case when all of the salary and bonus are redacted. For TI, the size of the certain tuples is decreasing in proportion to increasing redaction and ending at 0. In Figure 3.5(f), according to discussion in Section 3.8.4, the WHERE clause becomes a tautology when *salary* and *bonus* are redacted jointly. In this case, TC can capture the full semantics of the query no matter how much salary and bonus is removed. As expected, TI's result becomes worse when more salary and bonus is removed.

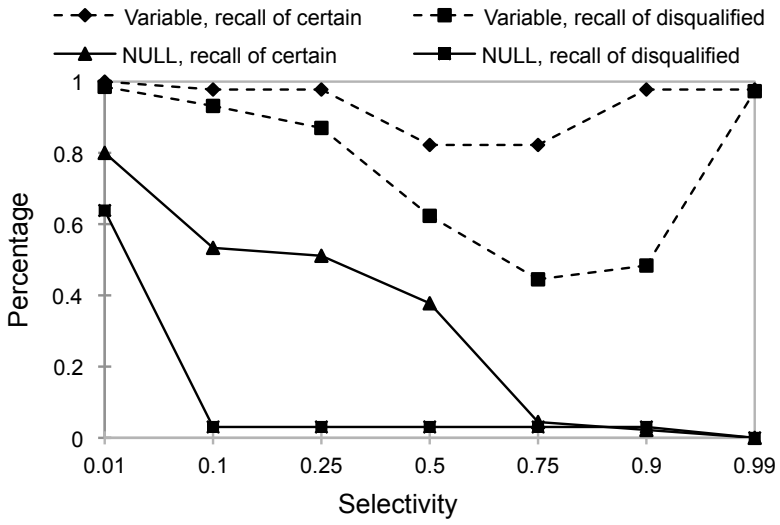


Figure 3.6: Compare the accuracy of query results between suppression with variables and suppression with NULLs, measured by the recall of certain and disqualified tuples.

Suppression using variables v. NULLs In our final experiments we apply redaction policies with a suppression function that uses NULL values instead of variables. Recall that using nulls in TI, called TI^{null}, has the same expressive power as Gadia's model [31] for incomplete temporal databases. Figure 3.6 shows the recall of certain and disqualified tuples on GetEmp2 (with condition on an early employee) compared with the variable solution. Variables significantly outperform NULLs. For example, with a selectivity of 25% the recall of certain tuples is 97% using variables, but just 56% using NULLs. This is because any two tuples with NULL on the join column will produce a possible output tuple. With distinct variable assignments, only identical variables will result in an output tuple.

3.10 Related Work

Retention policies and problems of expiring historical data have been studied in a variety of contexts. Garcia-Molina et al. considered expiring tuples from materialized views in a data warehouse [33]. An administrator can declaratively request to remove tuples from a view, and the system will remove as much information as possible as

long as it does not impact views referencing the original view. Toman proposed techniques for automatically expiring data in a historical data warehouse while preserving answers to a fixed set of queries [107]. Skyt et al. consider vacuuming a temporal database [98]. Policies remove entire tuples, and the authors are concerned with the correctness of vacuum specifications, and mitigating actions to handle queries referencing missing information. The above works differ from ours because they do not consider cell-level removal, do not view the resulting database as an incomplete history from which possible answers can be derived, and do not consider an audit log accompanying the history. Ataullah et al. [4] considered retention restrictions on complex business records, which they describe by logical views over relations. They define protective and destructive policies, and reduce a number of retention problems to well-studied relational view problems.

A number of authors have considered maintaining data integrity and preventing deletion in the context of auditing. Hasan and Winslett [43] considered the case when requested information is subject to a litigation hold and they addressed the threat of an untrustworthy process vacuuming expired records. Their solution uses write-once read-many (WORM) storage and extra auditing actions for enforcement of a litigation hold, instead of relying on a DBMS. In [44], they proposed a transaction log architecture to ensure that database contents are long-term immutable. Both of these solutions are complementary to our framework when considering the institution itself as an adversary. In a different setting, Perez and Moreau consider the problem of securing provenance-based audits [82] by protecting the integrity of provenance information. Fabbri et. al. [28] detect unauthorized access by re-executing a log of past operations. Encrypting audit logs has also been widely studied in the literature [94, 104, 110] with the goal of maintaining the confidentiality and integrity of log records.

Our redaction policies (especially when implemented logically) are related to fine-grained access control rules. Wang et al. [108] studied the correctness of query answers under cell-level access control policies, and made an important connection between that problem and models of incomplete information. To our knowledge there is little work on access control over time-varying data. Research into temporal access control models [5] refers to access rights that change over time, not the problem of negotiating access to data with a time dimension.

When computing possible tuples in TC, efficiently solving the satisfiability problem of conjunctive inequalities is essential. Generally, the complexity depends on the domain of variables (dense or sparse) which is determined by the corresponding column in the schema, supported operators ($=, <, >, \leq, \geq, \neq$), form of conditions ($X$ op Y, X op C or more general linear inequalities) and type of formulas (conjunctive or disjunctive). In [89], the authors proved the general satisfiability problem for conjunctive inequalities is NP-hard. Restricted versions, such as eliminating $\neq$ and only considering the real domain can be solved efficiently in linear time [40]. For disjunctive inequalities, Hochbaum [47] proved that even 2I-SAT is NP-complete, when considering linear inequalities. 2I-SAT only allows at most two inequalities per clause. The distinctness among variables in TC distinguishes our problem from all of the above.

The scheduling problem has a close relationship to our TC model. The "jobs" in the scheduling problem correspond to variables in TC. The distinctness among variables guarantees that each job will start at a different time on a single machine. $\text{TC}_{<}$ and $\text{TC}_{\neq}$ can be solved by efficient scheduling algorithms [59, 97]. It is possible that other scheduling solutions are applicable to TC and its variants.

CHAPTER 4

SHARING PRIVATE SYNTHETIC DATABASE

This chapter addresses the problem of sharing a synthetic database under differential privacy. Our database synthesis, though private, preserves the core performance metrics for a given query workload. The solution proposed is a model-based method, where we first select a model of database, then sanitize the statistics computed from the model, finally release synthetic databases that are sampled from the private statistics. To perturb the statistics, this chapter also describes the crucial extension of differential privacy to support multi-relation databases.

4.1 Introduction

Assessing the performance of database technologies depends critically on test databases and sample query workloads. A database vendor or researcher who has designed a novel database feature needs to evaluate the performance of her technology in the context of a real enterprise in order to measure performance gains. This applies broadly to new storage architectures, new query optimization strategies, new cardinality estimation methods, new physical or logical designs, new algorithms for automated index selection, etc.

This system evaluation would ideally be carried out using the actual data and query workloads used by the enterprise. Unfortunately, the actual data is often unavailable to the evaluator because privacy, security, and competitiveness concerns prevent the enterprise from releasing their data. The evaluator could resort to common benchmark databases (e.g. a TPC benchmark), which have been designed to

capture common properties of popular application domains. But because benchmarks target the common case, they often cannot reflect particular properties that may significantly impact performance for a given enterprise. Researchers have also proposed a number of database generation techniques [2, 8, 13, 48, 65, 106] that are able to create databases with specific characteristics. For example, when testing cardinality estimation methods, it is typically important to manipulate the skew of attribute distributions in test data. But without access to real databases and workloads, they can only guess at meaningful parameter settings for database generators. A final alternative is to employ techniques for synthesizing databases that match a given true database [2, 7, 27]. Unfortunately, none of these approaches provide a guarantee of privacy and, in fact, many of them produce output that can easily lead to serious privacy leaks.

The goal of our work is to safely support accurate performance analysis by potentially untrusted evaluators. We describe techniques for synthesizing, in a provably private manner, a relational database instance that matches the performance properties of the original database, especially with respect to a given target workload of SQL queries. The private synthetic data sets can be safely released to a vendor or researcher, and are designed to preserve core performance properties of queries such as IO counts, results sizes, and execution times.

Our approach is based on model-based database synthesis, as illustrated in Figure 4.1. We consider the *owner* of a sensitive database instance D, which conforms to schema S, along with a workload W containing queries commonly executed over the database. An untrusted *evaluator* would ideally like to carry out performance analysis using each of S, D, and W, but is prevented from doing so by privacy concerns. We obfuscate the schema by transforming S into an isomorphic schema S', and likewise transform W into W' by re-expressing queries in W in terms of the new schema S'.

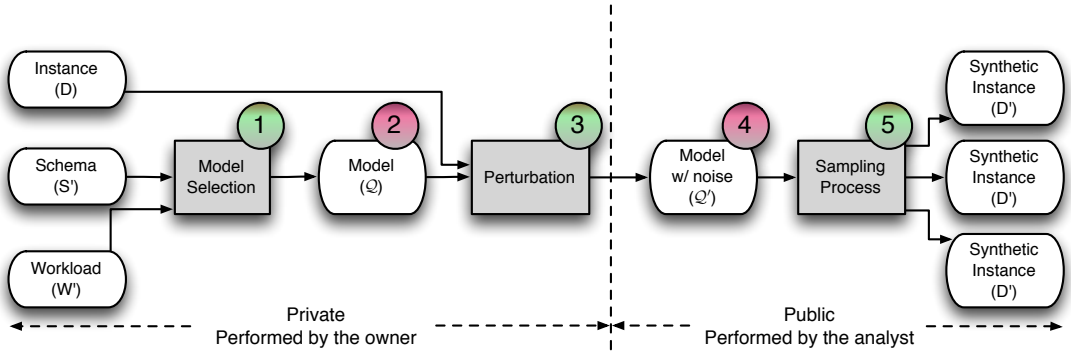


Figure 4.1: Our Approach: the *owner* selects (procedure of box 1, Section 4.3) a model $\mathcal{Q}$ (rounded box 2) given schema and workload. A model contains a set of carefully chosen queries, and their answers (statistics) can be calculated with instance (D). The owner now perturbs (procedure of box 3, Section 4.5) the statistics to get a differentially private $\mathcal{Q}'$ (rounded box 4). With the release of $\mathcal{Q}'$, the *analyst* can create/sample (procedure of box 5, Section 4.6) one or more synthetic instances.

We then provide a method for the owner to select, based on the schema and workload, a set of queries that serve as a model $\mathcal{Q}$ of the database D. Using this model and the dataset, a set of statistics are calculated and then perturbed so that it satisfies the formal standard of differential privacy. The perturbed results, $\mathcal{Q}'$, can be safely released to the evaluator and any computation using $\mathcal{Q}'$ will not weaken the privacy guarantee. Finally, the analyst, in possession of S', W', and $\mathcal{Q}'$, can generate a synthetic database instance consistent with the schema and statistics. There are typically many instances consistent with $\mathcal{Q}'$, so the analyst can generate many alternative database instances by sampling. An appealing by-product of our approach is that the analyst can also choose to generate scaled-up synthetic databases to evaluate performance on larger, statistically-similar instances.

Contributions

We achieve the goals of untrusted system evaluation through the following contributions. First, we extend differential privacy to multiple tables, re-defining the concept of neighboring databases and sensitivity. This is a crucial extension for our framework and also useful beyond the present work. Next we propose a novel algorithm for selecting the queries that constitute the model $\mathcal{Q}$, where we must balance

descriptive power with accuracy achievable under the privacy condition. After privately estimating the selected model statistics to produce $\mathcal{Q}'$ we then propose an efficient method for consistently sampling from $\mathcal{Q}'$ to generate a privacy-preserving synthetic instance of the database. Lastly, we assess the accuracy of our techniques for a range of performance metrics. We compare the value of these metrics for the true database, synthetic data generated from non-private models, and synthetic data generated from private models. We conclude that the distortion due to privacy is modest and that important performance properties are retained in the output.

4.2 Preliminaries

In this section we describe our data model, queries, the definition of differential privacy, and the primary privacy mechanism we apply.

4.2.1 Data model and queries

We consider a database D that is an instance of schema $S = \{R_1, R_2, \ldots\}$. System evaluation is performed with respect to a workload of queries W consisting of SQL queries. A table $R = (A_1, A_2, \ldots)$ in S contains *key* attributes and *non-key* attributes, where the key attributes may be primary or foreign keys. Throughout the chapter, we focus on workload queries involving joins only on key attributes. This assumption is also accepted by the literature (e.g. [2]) and it actually covers a wide range of applications, including TPC-H benchmark. However, we claim that our privacy definition and mechanism is not restricted to such queries. We represent the schema S as a directed graph G_S, where each table is then a node and edges are drawn from R_i to R_j when R_j contains a foreign key reference to a key attribute in R_i. An example schema graph for TPC-H is shown in Figure 4.2, containing relations R(region), N(nation), C(customer), O(orders), L(lineitem), P(part), S(supplier) and PS(partsupp). We limit our attention to schemas with acyclic schema graphs.

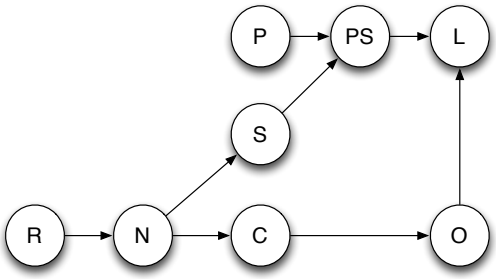


Figure 4.2: The schema of TPC-H represented as a directed graph.

A counting query q is an aggregate query that returns the number of tuples satisfying one or more predicates. A counting query may involve a single table or multiple tables joined by their keys and foreign-keys. We refer to the relationship among tables involved in the query as its *signature*, denoted by $v(q)$. Counting queries are written in relational algebra, as in the following examples:

$$q_1 \quad : \quad |\sigma_{\mathsf{C}.gender=M}(\mathsf{C})|$$

$$q_2 \quad : \quad |\sigma_{\mathsf{C}.gender=M}(\mathsf{C} \bowtie \mathsf{O})|$$

These two counting queries return the number of male customers and the number of orders from male customers, respectively. The signature of q_1 is $v(q_1) = \mathsf{C}$ and the signature of q_2 is $v(q_2) = \mathsf{C} \bowtie \mathsf{O}$.

The model $\mathcal{Q}$ of the owner's database, shown in Fig. 4.1 and described in detail in the next section, is defined by a set of counting queries derived from the workload. We refer to this set of counting queries as the *model queries*. Note that while the model queries are restricted to counting queries, the workload may contain more general queries.

4.2.2 The differential privacy guarantee

An algorithm is differentially private if its output is statistically close on two database inputs that differ by one record. Recall in Section 2.3, we introduce the

formal definition of differential privacy and basic methods that implement it, such as Laplace mechanism. Achieving differential privacy requires perturbing statistics computed from the true database. This perturbation protects against disclosures that can result from releasing exact statistics about the original database, as is done by existing database synthesis techniques [2, 7, 27].

In Section 4.4 we extend differential privacy to complex schemas with multiple tables by focusing on a protected entity and the entity's relationships. However, we note that even under this extension, differential privacy does not offer protection for the population. In our setting, the differential guarantee (which applies to the model $\mathcal{Q}$ of D) means that we reveal very little about protected entities and their relationships. But it does not prevent the release of accurate aggregates for the population (and in fact we require reasonably accurate aggregates in order to capture the properties of D). In some settings, these aggregate query answers may not be acceptable to release. For example, the average revenue for a company or the total number of customers may be sensitive values, even when the individual records contributing to these aggregates remain protected. In domains where population aggregates are highly sensitive, accurate and private database synthesis is likely to be impossible. Nevertheless, we believe there are a wide range of applications in which the primarily concern is the sensitivity of individual entities for which our techniques provide strong privacy. Practical examples are requirements of working with medical information [29], location data [18] and network traces [72].

The models of the database we consider are defined (in the next section) by sets of counting queries over D. To release a differentially-private model to the evaluator, we must produce private answers to a large and potentially complex set of counting queries. The standard mechanisms (the Laplace for ϵ-differential privacy and Gaussian (ϵ, δ)-differential privacy) are quite effective at answering single queries, but can be highly sub-optimal for the large sets of queries we consider. Intuitively, one rea-

son for this is that the counting queries in our models may overlap, leading to high sensitivity and high per-query error.

Improved methods for answering sets of counting queries have received considerable attention from the research community recently [19, 21, 42, 46, 63, 115–117]. Our goal is framework for database generation that is agnostic to any particular privacy mechanism. Thus choose to adapt the recent work by Li et al [64], based on the matrix mechanism [63], for answering multiple linear counting queries with low error. This technique offers an adaptive mechanism which adds noise customized to the set of counting queries required by the model. The adaptive method works best for (ϵ, δ)-differential privacy (achieving error rates that are very close to a theoretical lower bound for mechanisms of this form) and we therefore focus our experiments on the mechanism satisfying this relaxed version of differential privacy.

We emphasize that our framework is largely independent of a particular mechanism used to derive the private model. This means that, in the future, better utility could be achieved using our framework as privacy techniques advance.

4.3 Deriving a model from a query workload

In this section we describe the process for deriving a statistical model of the input database, and in particular, a model which is specialized to a given set of workload queries. The challenge is selecting a model that captures properties of the database relevant to performance evaluation while at the same time allowing for accurate release under differential privacy. We restrict our attention to classical relational database systems and workloads of SQL queries.

4.3.1 Extracting counting queries

The selected model will be defined by a set of counting queries. We select counting queries relevant to a given workload of SQL queries by considering intermediate

operations in the query evaluation process, similar to Arasu et al [2]. Ideally, the synthetic database sampled should produce similar executions when running each workload query. The cardinality of each intermediate operator output are called an *intermediate count*. Since a modern query optimizer uses table statistics to generate query plans, if our model gathers all the intermediate counts of query trees, i.e., the size of intermediate results on each node of the query tree, the optimizer will utilize the same table statistics as the original databases to produce query plans.

The intermediate counts are represented as counting queries, and they are independent of the data instance, DBMS and physical organization of data. Let w be a single workload query. $\Gamma(w)$ is the set of statistics (counting queries) that can be extracted from any possible query tree of w. With $v(w)$ as the signature of w and $|v(w)|$ as the number of tables in the signature, we can describe $\Gamma(w)$ as follows:

$$\Gamma(w) = \{\Gamma_0(w), \Gamma_1(w), \Gamma_2(w), \ldots, \Gamma_{|v(w)|}(w)\}$$

Each $\Gamma_i(w)$ is the set of all counting queries over an i-way join of a subset of tables in $v(w)$. In fact, each item in $\Gamma_i(w)$ represents the size of the intermediate result of a node that involves an i-way join, thus each counting query can be mapped to a node in some query tree. In particular, $\Gamma_0(w)$ contains counting queries for the size of each table in $v(w)$. For a multi-query workload W, we let $m = max_{w \in W}(|v(w)|)$, and $\Gamma_i(W) = \bigcup_{w \in W} \Gamma_i(w)$, and define:

$$\Gamma(W) = \bigcup_{i=0,1,..,m} \Gamma_i(W)$$

Example 4.1. Assume a workload $W = \{w_1, w_2\}$ consisting of two queries:

$$w_1 \quad : \quad \sigma_{\mathsf{C}.gender=M \wedge \mathsf{O}.year=2010}(\mathsf{C} \bowtie \mathsf{O})$$

$$w_2 \quad : \quad \sigma_{\mathsf{C}.age=40}(\mathsf{C})$$

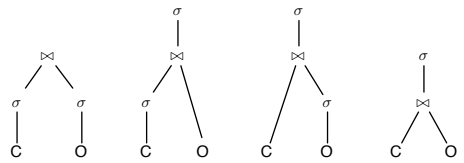


Figure 4.3: Possible query trees for $\sigma_{\mathsf{C}.gender=M \wedge \mathsf{O}.amount>100}(\mathsf{C} \bowtie \mathsf{O})$

$\Gamma(w_1)$ includes intermediate counts up to the 2-way join and $\Gamma(w_2)$ includes counts over a single table. The set of intermediate counts of w_1 is derived from the four possible query trees (Figure 4.3). Thus, $\Gamma(W)$ is the union of following:

$$
\begin{aligned}
\Gamma_0(W) &: |\mathsf{C}|, |\mathsf{O}| \\
\Gamma_1(W) &: |\sigma_{\mathsf{C}.gender=M}(\mathsf{C})|, |\sigma_{\mathsf{O}.year=2010}(\mathsf{O})|, \\
&\quad |\sigma_{\mathsf{C}.age=40}(\mathsf{O})| \\
\Gamma_2(W) &: |\sigma_{\mathsf{C}.gender=M}(\mathsf{C} \bowtie \mathsf{O})|, |\sigma_{\mathsf{O}.year=2010}(\mathsf{C} \bowtie \mathsf{O})|, \\
&\quad |\sigma_{\mathsf{C}.gender=M \wedge \mathsf{O}.year=2010}(\mathsf{C} \bowtie \mathsf{O})|
\end{aligned}
$$

To select a good query plan, the query optimizer will estimate the number of rows retrieved by the query using stored statistics on the data distribution. Although we do not directly measure the data distribution on all attributes, the counting queries we extract as model statistics represent a rough approximation of this, namely those statistics relevant to the queries in the workload of interest.

4.3.2 A spectrum of models

Next we define a spectrum of models, each derived from the workload. While the most descriptive model would likely be preferred in the absence of privacy concerns, in our setting, a more descriptive model can ultimately be less effective because more distortion must be applied to satisfy the privacy condition.

The most descriptive model is a *Saturated Model* **(SM)** that contains all intermediate counts (counting queries) of any possible query tree. SM gathers the most

information from the workload, but its size grows quickly as the workload becomes larger, particularly when multiway joins are involved. Moreover, SM will typically contain many related counting queries, resulting in high sensitivity, and requiring significant noise in the perturbation step. Therefore, we identify a number of simpler models. The idea is to quantify proper correlation among tables using intermediate counts, which is generally identified as *Correlation of i-Table Model*, shortened as **CiTM**, where $i \in \mathbb{N}$.

The **C1TM** model considers just intermediate counts within a single table, which are the set of all counting queries corresponding to leaf nodes in a query tree. The **C2TM** model includes up to 2-way cross-table correlations, consisting of the intermediate counts in a query tree from the leaves and their parents. In general, there exist models that include up to the i-way cross-table relationships. For comparison purposes, we also consider a *Null Model* **(NM)**, reflecting only of the size of each relation and containing nothing about the workload. For a set of workload queries W, these models can be formally described as follows:

$$\begin{aligned} \mathcal{Q}_{\mathrm{SM}} &= \Gamma(W) \\ \mathcal{Q}_{\mathrm{C}i\mathrm{TM}} &= \Gamma_0(W) \cup \Gamma_1(W) \cup \ldots \cup \Gamma_i(W) \\ \mathcal{Q}_{\mathrm{NM}} &= \Gamma_0(W) \end{aligned}$$

With $\Gamma(W)$, we are able to define *a family of models*, by putting together arbitrary $\Gamma_i(W)$. Selecting a model is complex because greater descriptive power in a model generally means it has a higher privacy cost and therefore demands greater perturbation for a fixed setting of the privacy parameters. We will show in the following sections that the amount of perturbation required by a model can be calculated directly and we evaluate the impact of distortion on performance testing in the experimental evaluation.

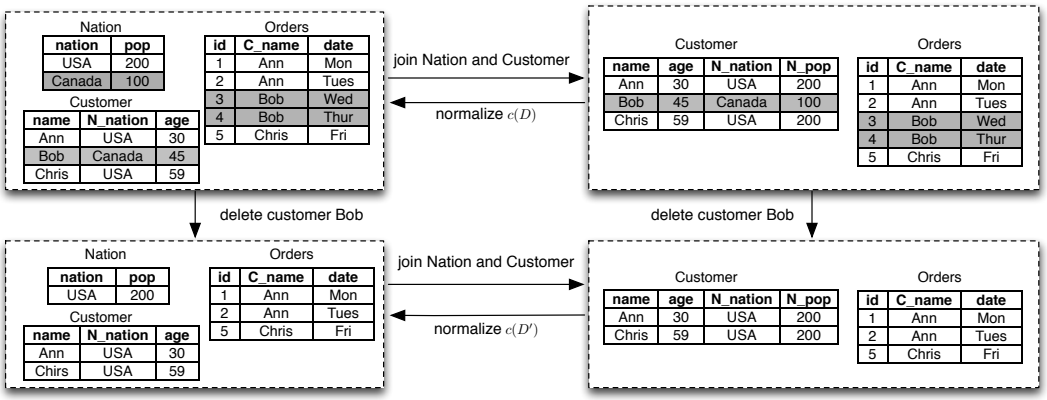


Figure 4.4: An example of neighboring multi-relation databases for schema $S = \{\mathsf{N}, \mathsf{C}, \mathsf{O}\}$. D and D' are neighbors because collapsed instances $c(D)$ and $c(D')$ are neighbors where $c(D')$ is generated by a cascading deletion of customer Bob from $c(D)$. Note that Canada is missing from D' as Bob is the only customer from Canada.

4.4 Differential privacy for multiple-relation databases

In this section we extend the standard definition of differential privacy from a single relation to multiple relations. The original differential guarantee protects individuals in a single-relation database by requiring statistically close outputs on neighboring databases that differ on a single tuple. Using such a notion of neighboring databases in the context of a multi-relation database is insufficient because an individual's sensitive information will be represented in multiple tables. Considering TPC-H as an example, each customer is associated with multiple orders. Under single-table differential privacy, a query reporting the average order amount for a customer may reveal the fact that a customer has an extremely high number of orders due to insufficient noise. A similar issue has been identified by Kifer et al [57]. However, since a general schema may have complicated relationships among relations, defining differential privacy for multiple relations is not straightforward. We will show below that even the calculation of query sensitivity requires careful consideration. The PINQ system [74] also deals with this problem, but instead of proposing a direct solution, it uses a modified non-standard semantics of join which is not applicable in our scenario.

In the following, we first generalize the notion of neighboring databases, focusing on a single *protected entity* but accounting for tables related by key/foreign-key rela-

tionships. We then discuss the calculation of query sensitivity and the calculation of sensitivity for the queries that make up a model.

4.4.1 Multi-relation neighboring databases

We assume that a single table is identified as the primary protected entity in the schema. In TPC-H , we choose the customer table as the protected entity (relation C). We then seek to protect each customer's data, including their participation across multiple relations connected by key/foreign-key constraints. To do so, we consider the following categorization of tables based on a schema graph.

1) Relations that are ancestors of the protected entity represent properties of the entity that happen to be stored in separate relations. These should be protected along with attributes in the tuples of the protected entity table. For example, table N is an ancestor of C in the graph defined by the TPC-H schema and stores a customers' nationality, which should be protected.

2) Relations that are descendants of the protected entity represent a set-valued property of the entity that should be protected. For example, O and L are descendants of C. In the order table O, there are multiple orders associated with each customer which deserve protection. Removing one customer should result in a *cascading deletion* of tuples from descendant relations, e.g., deleting the multiple associated orders from O.

3) Ancestors of the protected entity's descendants (but not direct ancestors) can be viewed as properties of the items represented by entity's descendants. E.g., when protecting lineitem L as a set-valued property of customers, each lineitem's supplier, stored in S, should also be protected.

To formalize neighboring databases in multiple relations, we introduce a partially denormalized version of D, $c(D)$, generated by repeatedly performing pairwise joins on key and foreign keys until the database contains only the protected relation R

and its descendants (see Figure 4.4 for an example). We say $c(D)$ is *reversible*, if the normalization of $c(D)$ results in the original D. Consider a relation X's primary key is referenced by Y's foreign key, $X \to Y$, we say this relationship satisfies an *inclusion constraint* if each of X's keys are referenced at least once in Y. If inclusion constraints are held among all of the pairs of tables that are being joined during the creation of $c(D)$, reversibility is then guaranteed, giving us the ability of rebuilding the original database.

Definition 4.2 (Neighboring databases)**.** Let D and D' be instances of schema S such that their partially denormalized versions $c(D)$ and $c(D')$ are reversible. D and D' are neighbors if $c(D)$ is generated by cascade deleting some tuple in $c(D)$ from database $c(D')$, or vice versa.

Definition 4.2 completes our definition of neighboring databases for multi-relation databases, where denormalized databases help to take care of cascading deletion starting from the protected entity, and reversibility helps to maintain consistency on all other tables that are not involved in the cascading process.

Example 4.3. Suppose we have a simplified TPC-H schema $S = \{\mathsf{N}, \mathsf{C}, \mathsf{O}\}$ with $\mathsf{N} \to \mathsf{C} \to \mathsf{O}$. Figure 4.4 demonstrates two example neighboring databases and their collapsed versions, and the relationship between these two versions.

Remark. The assumption of reversibility simplifies the definition of neighboring databases, but is not a requirement.

4.4.2 Query sensitivity

We turn next to computing the sensitivity of queries, which is the maximum change in a query answer for two neighboring databases. We first calculate Δ_q under single table differential privacy by viewing signature $v(q)$ as a virtually materialized single table and therefore the difference between neighbors is one. Under multi-

relation differential privacy, $v(q)$ in neighbors can differ by more than one, thus the sensitivity of q should be augmented a factor of that difference (the *df* value):

$$\Delta_q \cdot df(v(q)) \tag{4.1}$$

From this point forward, without additional notation, Δ always refers to the sensitivity in multi-relation differential privacy, as single-relation differential privacy is just a special case with *df* value equal to 1 for every table.

The key of computing sensitivity under multi-relation differential privacy is to calculate the *df* value. We begin by considering a single-table counting query, where the signature is always a single relation, say X. It is obvious that $df(X)$ is one if X is the protected entity table, but for other tables this number is not constant, as one customer could potentially match as many orders as possible so *df* value of O table could be as large as its size.

We address this issue by assuming a bound on the join frequency across tables. We refer to this as a *propagation constraint*, $\mathcal{K}(X, Y)$, defined as the maximum number of times that each primary key in table X can be referenced in table Y for the key/foreign-key relationship $X \to Y$. With a fixed schema, the propagation constraint is the only variation to decide a query's sensitivity. A given propagation constraint $\mathcal{K}$ indicates that differential privacy *fully* protects the individual/entity that has join frequency smaller than $\mathcal{K}$. Those with frequency larger than $\mathcal{K}$, will be *partially* protected. Therefore, with consideration of utility, we also choose $\mathcal{K}$ as large as possible. When $\mathcal{K}$ is equal to the maximum join frequency, all tuples in X are protected.

Algorithm 4.1 computes the *df* value for each table, assuming R is the protected entity for schema graph G_S. We use $desc(R)$ to refer to the set of all descendants of R.

Algorithm 4.1 Compute df value

1: **for** X in topological order of G_S **do**
2: **if** $X == R$ **then** $df(X) = 1$
3: **else if** $X \in desc(R)$ **then**
4: $df(X) = \sum_{Y \to X} \mathcal{K}(Y, X) df(Y)$
5: **else** $df(X) = 0$
6: **for** X in reverse topological order of G_S **do**
7: $df(X) = df(X) + \sum_{X \to Y} [df(Y) - \mathcal{K}(X, Y) df(X)]$
return all df values

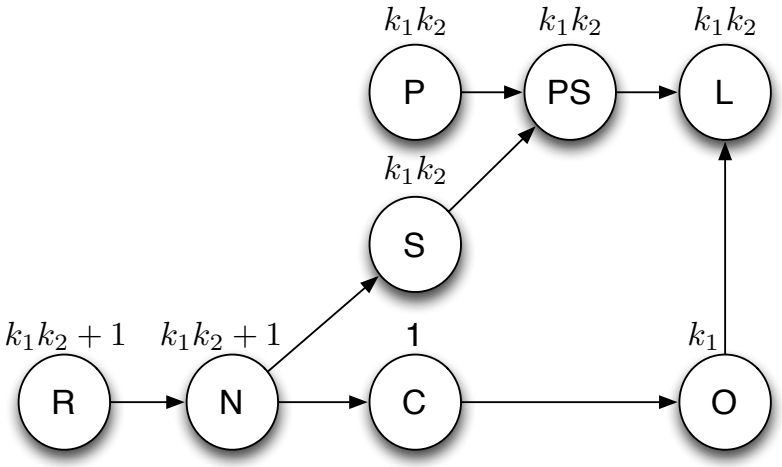


Figure 4.5: Difference (df value) between neighboring TPC-H instances.

Example 4.4. Let C be the protected table and $\mathcal{K}(\mathsf{C}, \mathsf{O}) = k_1$, $\mathcal{K}(\mathsf{O}, \mathsf{L}) = k_2$. As shown in Fig. 4.5, $df(\mathsf{C}) = 1$. If each customer associates with at most k_1 orders, $df(\mathsf{O})$ is $1 * k_1 = k_1$. Similarly, $df(\mathsf{L}) = k_1k_2$. Then we begin the round of reverse topological order. We pick the PS table, since it is the only table with all of its children (L) computed. If k_1k_2 lineitems are deleted in L, there are at most k_1k_2 tuples deleted in PS (an upper bound for all cases). Thus, $df(\mathsf{PS}) = k_1k_2$. After that, we consider P and S. $df(\mathsf{N}) = df(\mathsf{S}) + df(\mathsf{C})$ because deleted tuples in S and C could refer to different nations. At last, we calculate $df(\mathsf{R})$.

Now we consider the case that a counting query's signature involves joins of multiple tables. As the join operation propagates the primary-key table into the foreign-key table, the maximum difference after the join is just the df value of the foreign key table, given by the following equation for a 2-way join:

$$df(X \bowtie Y) = df(Y) \quad \text{if } X \to Y$$

For example, in Figure 4.5, $df(\mathsf{N} \bowtie \mathsf{C}) = df(\mathsf{C}) = 1$, since the removal of one tuple in the customer table will cause at most one nation to be deleted in the nation table. We do not consider deletions propagated from S, because they do not influence the join on N and C. Generally, if there are multiple tables joined (i.e. more than two) in the signature of a query, we repeatedly apply this equation, and the *df* value is always equal to the last referenced table if there is only one such table. If the signature of a query is not sequential (e.g., $\mathsf{C} \bowtie \mathsf{O} \bowtie \mathsf{L}$) or snowflake (e.g., $(\mathsf{P} \bowtie (\mathsf{S} \bowtie \mathsf{PS}))$, its overall *df* is the sum of *df* values on each of last referenced table, such as $df(\mathsf{S} \bowtie \mathsf{N} \bowtie \mathsf{C}) = df(\mathsf{S}) + df(\mathsf{C})$. Moreover, the definition of neighboring databases proposed in Section 4.4.1 is indeed independent of queries, which means with proper modification to the methods discussed above (e.g., knowing propagation factors for non-key attributes), we can calculate the sensitivity for queries that beyond key-key joins. We omit them from the discussion here.

4.5 Model Perturbation

Given a selected model $\mathcal{Q}$, our next goal is to perturb the true query answers of the model to satisfy multi-relational differential privacy. A simple approach is to calculate the sensitivity of the whole model and then add noise calibrated to the sensitivity. However, in the case of multi-relations, this method would add more noise than strictly necessary to satisfy the privacy criterion, and would hurt utility. Instead we invoke privacy mechanisms multiple times, the challenges are to generate an optimal mechanism composition and budget allocation, and effectively deal with data representation for multi-relation correlations. In this section, we propose a framework for resolving these challenges.

4.5.1 General framework for working with multi-relations

We apply a *data vector* based representation for databases and queries to help deploy the perturbation process. In our framework, each table is encoded as a data vector. A data vector $\mathbf{x}$ consists of *cell counts*, which are the counts of tuples that satisfy a set of disjoint *cell conditions* (Figure 4.6(b)). Essentially, a data vector is similar to a multi-dimensional histogram, containing a set of dimensions, e.g., $dim(\mathbf{x}) = \{age, gender\}$. Note that the dimensions do not need to contain all attributes of a table. Using data vector $\mathbf{x}$, a counting query q can be expressed as $|\mathbf{x}|$ coefficients and all counting queries are combined as a *query matrix* $\mathbf{Q}$ with each row as one query. E.g., $\mathbf{Q}$ (Figure 4.6(c)) is the query matrix containing the three counting queries of Figure 4.6(a) based on $\mathbf{x}$ in Figure 4.6(b). The true answers to the counting queries are computed as the matrix product of $\mathbf{Q}$ and $\mathbf{x}$. Thus, the *Gaussian mechanism* for the single-table database, which adds Gaussian noise calibrated to the L_2 sensitivity (noted as Δ) to achieve (ϵ, δ)-differential privacy [23], can be defined as:

Definition 4.5 (Gaussian Mechanism)**.** Assume $\mathbf{Q}$ contains d queries, the following randomized algorithm $\mathcal{G}$ provides (ϵ, δ)-differential privacy on input database D. Here the sensitivity $\Delta_{\mathbf{Q}}$ is equal to the maximum L_2 norm of a column.

$$\mathcal{G}(\mathbf{Q}, D) = \mathbf{Q}(D) + \text{Normal}(\frac{\Delta_{\mathbf{Q}}\sqrt{2\ln(2/\delta)}}{\epsilon})^d$$

With multiple relations, it is not possible to construct a single data vector and format all the model queries. Instead, the general framework is that we encode a multi-relation database into a set of data vectors $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$ and thus a model $\mathcal{Q}$ can be represented as n query matrices $\mathcal{Q} = \{\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_n\}$. Since there is no direct privacy mechanism designed for multiple relations, we invoke a single-table mechanism multiple times under *mechanism composition* with a properly distributed

the privacy budget. We call such a mechanism a *unit mechanism*. A simple example is to set the unit mechanism to be Gaussian mechanism and run it on each $(\mathbf{x}, \mathbf{Q})$ pairs, under both sequential and parallel composition rules.

The first problem of this composition framework is the choice of data vectors because there is more than one way to represent $\mathcal{X}$. Although we always have logically equivalent representations of the model queries, the choice of $\mathcal{X}$ can impact answer consistency. Consider a model with two counting queries $q_1 = |\mathsf{O}|$ and $q_2 = |\mathsf{C} \bowtie \mathsf{O}|$, represented by two different data vectors encoding O and $\mathsf{C} \bowtie \mathsf{O}$ without common dimensions. When applying a unit mechanism on each of them, independent noise will be added and the perturbed answers will not necessarily be the same. This is an *inconsistent* state because these two queries are actually equivalent if a foreign key constraint holds. As the perturbation of each $(\mathbf{x}, \mathbf{Q})$ is independent, the data vector representation does not depend on the unit mechanism used in the framework.

The other problem is to distribute the privacy budget efficiently. Data vectors may come from tables with different *df* values in terms of sensitivity calculation (Section 4.4.2), thus simply splitting the privacy budget evenly among invocations does not always give the minimal error under composition. Other than the choice of data vector representation, each choice of unit mechanism needs a particular budget allocation plan to optimize the perturbation error. For example, the Laplace and Gaussian mechanism have different budget allocation in our framework.

4.5.2 Choice of data vectors

Inconsistency from noisy answers arises because there is shared information among data vectors. In the example above, two data vectors share common total counts. The solution is to build data vectors that always contain all key/foreign-key relationships of ancestors. We refer to them as *denormalized data vectors*, where attributes in ancestors are viewed as simple properties of the current relation. For example, for

q_1: number of customers
q_2: number of male customers
q_3: difference between young and old customers

(a) Counting queries

$\mathbf{x}$	cell condition
2	age≤40, gender=M
1	age≤40, gender=F
2	age>40, gender=M
1	age>40, gender=F

(b) data vector $\mathbf{x}$ of customer table

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 0 \end{bmatrix}$$

(c) The counting queries from (a) represented as a matrix $\mathbf{Q}$ based on $\mathbf{x}$. The answers to $\mathbf{Q}$ are $\mathbf{Qx}$.

Figure 4.6: An example of counting queries and a data vector.

relation O, with ancestors R, N and C, we build a data vector based on the joined result of $\mathsf{R} \bowtie \mathsf{N} \bowtie \mathsf{C} \bowtie \mathsf{O}$. We do this for each relation in the database and now the two queries in the example above will be represented using the data vector on O and consistency is maintained after perturbation. Under this scheme, when merging two data vectors, correspondent model queries can be transformed automatically, essentially summing over the extra dimensions in the expanded data vector.

Example 4.6. Consider the saturated model for workload queries W in Example 4.1 (Section 4.3.1). A consistent representation can be built with two data vectors $\mathbf{x}_\mathsf{C}$ and $\mathbf{x}_\mathsf{O}$, where $\dim(\mathbf{x}_\mathsf{C}) = \{\mathsf{C}.age, \mathsf{C}.gender\}$ and $\dim(\mathbf{x}_\mathsf{O}) = \{\mathsf{C}.gender, \mathsf{O}.year\}$. These two vectors contain all ancestor relationships, but skip unnecessary columns to minimize the size of the vectors, e.g., $\mathbf{x}_\mathsf{O}$ does not include $\mathsf{C}.age$ as no model queries related to $\mathbf{x}_\mathsf{O}$ apply conditions on it. For model query transformation, look at $|\sigma_{\mathsf{C}.gender=M}(\mathsf{C})|$, a model query in the C1TM model. By introducing $\mathbf{x}_\mathsf{C}$, it will be rewritten to sum up all male ages in the $\mathbf{x}_\mathsf{C}$, that is $|\sigma_{\mathsf{C}.gender=M,\mathsf{C}.age=*}(\mathsf{C})|$.

4.5.3 Minimizing perturbation error

Now we state our algorithm for budget allocation. A standard choice for the unit mechanism would be Laplace or Gaussian mechanism, both of which can fit

well in our framework when finding a best budget distribution plan is not difficult. To illustrate that our framework is independent of unit mechanisms, we employ the more advanced matrix mechanism [63]. Although it requires more dedicated design for budget allocation, we can reach much lower perturbation error. (In fact, the allocation algorithm for the matrix mechanism is an extended version of the allocation for Gaussian mechanism.)

4.5.3.1 The matrix mechanism

Under single-relation differential privacy, we can formally define the matrix mechanism as follows, where the key difference is that a new query set (the strategy, $\mathbf{A}$) is answered with the Gaussian mechanism and then the desired queries $\mathbf{Q}$ are derived from it:

Definition 4.7 (Matrix Mechanism)**.** [63] Let $\mathbf{A}$ be a query strategy matrix and $\mathbf{A}^+ = (\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}^t$, the pseudo-inverse of $\mathbf{A}$. The randomized algorithm $\mathcal{M}_\mathbf{A}$ offers (ϵ, δ)-differential privacy.

$$\mathcal{M}_\mathbf{A}(\mathbf{Q}, \mathbf{x}) = \mathbf{Q}\mathbf{A}^+\mathcal{G}(\mathbf{A}, \mathbf{x})$$

Intuitively, answering the strategy queries privately and then deriving the desired workload queries leads to greater accuracy when the workload queries have high sensitivity caused by many overlapping queries. The error of query estimates in the matrix mechanism is measured by the mean squared error, determined by $\mathbf{Q}$ and strategy $\mathbf{A}$ (independent of $\mathbf{x}$). The total error is given by the following equation:

$$\textsc{Err}(\mathbf{Q}, \mathbf{A}) \quad = \quad \frac{2\ln(2/\delta)}{\epsilon^2}\Delta^2_\mathbf{A}\ trace(\mathbf{Q}(\mathbf{A}^t\mathbf{A})^{-1}\mathbf{Q}^t) \tag{4.2}$$

The main challenge of the matrix mechanism is choosing a good strategy for the given queries $\mathbf{Q}$ and we rely on the algorithm in [64] to compute an approximately

optimal strategy for any given $\mathbf{Q}$. So in our multi-relation framework, multiple runs of matrix mechanism will need a series of strategies $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$ matched with data vectors $\mathcal{X}$ and $\mathcal{Q}$.

4.5.3.2 Sensitivity and composition rules

The sensitivity for a single $\mathbf{Q}$ or strategy matrix $\mathbf{A}$ is its maximum L_2 norm of a column, multiplied by the *df* value of the query signature. In general, the total sensitivity of multiple matrices may not be equal to the summation of each of them, i.e. $\Delta^2_{\mathcal{Q}} \leq \sum_{\mathbf{Q}\in\mathcal{Q}} \Delta^2_{\mathbf{Q}}$. This is due to the possible correlation among query matrices. In fact, calculation of the exact sensitivity relies on searching for a proper series of columns across each query matrix that maximize the sum of the square of L_2 norms. We omit the detailed discussion on sensitivity computation here, as we are always safe to use the upper bound as the sensitivity. In addition, in matrix mechanism, an optimal strategy matrix will always maximize the L_2 norm on each column [64], meaning all the columns have the same norm. Thus, each matrix contributes its full ability in the overall sensitivity of $\mathcal{A}$, which means it reaches the upper bound $\Delta^2_{\mathcal{A}} = \sum_i \Delta^2_{\mathbf{A}_i}$. However, note this equation does not hold for a general case of multiple query matrices.

An important part of our framework is to have certain composition rules for the unit mechanisms. We use the following sequential and parallel composition rules, originally proposed for the Laplace and Gaussian mechanisms [23, 73, 74], also apply to the matrix mechanism, for the first time.

Proposition 4.1 (Sequential composition)**.** *If each matrix mechanism $\mathcal{M}_{\mathbf{A}_i}$, operating on workload $\mathbf{Q}_i$ and data vector $\mathbf{x}_i$, provides (ϵ_i, δ_i)-differential privacy, sequential application of $\mathcal{M}_{\mathbf{A}_i}$ on each workload in $\mathcal{Q}$ satisfies $(\sum \epsilon_i, \sum \delta_i)$-differential privacy.*

Proposition 4.2 (Parallel composition)**.** *If each matrix mechanism $\mathcal{M}_{\mathbf{A}_i}$ uses the overall sensitivity for all strategy matrices $\Delta_{\mathcal{A}}$ to answer $\mathbf{Q}_i$ over $\mathbf{x}_i$, combination of all $\mathcal{M}_{\mathbf{A}_i}$ satisfies (ϵ, δ)-differential privacy.*

Using these two composition rules, we partition $\mathcal{Q}$ into multiple disjoint subsets/groups, such that the union of these sets equals $\mathcal{Q}$, and then apply Proposition 4.2 inside each group and Proposition 4.1 across groups. Note that $\mathcal{A}$ is also partitioned in the same way. All strategy matrices in one group will get a unified privacy budget assigned to that group. Let $\mathcal{I}$ be the set of all partitions for $\mathcal{Q}$. Two extreme partitions are the fully split one I_s with only single-sized groups (applying Proposition 4.1 only) and the fully joined one I_j with one group that contains all query matrices (applying Proposition 4.2 only).

4.5.3.3 Error for a partition

The total error of applying privacy mechanism $\mathcal{M}$ on partition I is the sum of errors from each group $g \in I$. Let $\textsc{ErrG}_{\mathcal{M}}(g, \epsilon_g, \delta_g)$ be the error of group g given privacy budget ϵ_g and δ_g. So the minimum total error of partition I, $\textsc{MinErr}_{\mathcal{M}}(I)$, is

$$\textsc{MinErr}_{\mathcal{M}}(I) = \min \sum_{\text{group } g \in I} \textsc{ErrG}_{\mathcal{M}}(g, \epsilon_g, \delta_g) \tag{4.3}$$

From Equation (4.2), the total error of applying matrix mechanism on one query matrix is $\textsc{Err}_{\mathcal{M}} = 2\ln(2/\delta)/\epsilon^2 \Delta^2_{\mathbf{A}} b$, where the trace value $b = \text{tr}(\mathbf{Q}(\mathbf{A}^t\mathbf{A})^{-1}\mathbf{Q}^t)$. Let $\Delta_{g(\mathcal{A})}$ be the sensitivity of group g's strategy matrices.

$$\begin{aligned} \textsc{ErrG}_{\mathcal{M}}(g, \epsilon_g, \delta_g) &= \sum_{i \in g} \frac{2\ln(2/\delta_g)}{\epsilon_g^2} \Delta^2_{g(\mathcal{A})}\, b_i \\ &= \frac{2\ln(2/\delta_g)}{\epsilon_g^2} (\sum_{i \in g} \Delta^2_{\mathbf{A}_i})(\sum_{i \in g} b_i) \end{aligned} \tag{4.4}$$

To calculate $\textsc{MinErr}_{\mathcal{M}}(I)$, we apply Lagrange multiplier to solve the optimization problem with objective function Equation (4.3) and two equality constraints $\sum_g \epsilon_g = \epsilon$ and $\sum_g \delta_g = \delta$, which gives us the following results:

$$\textsc{MinErr}_{\mathcal{M}}(I) \;=\; \frac{2}{\epsilon^2}\left(\sum_{\text{group } g\in I} \sqrt[3]{\ln(\frac{2}{\delta_g})\cdot b_g c_g}\right)^3 \tag{4.5}$$

Here, the group trace value, b_g, is defined as $\sum_{\mathbf{Q}_i,\mathbf{A}_i\in g} \text{tr}(\mathbf{Q}_i\,(\mathbf{A}_i{}^t\mathbf{A}_i)^{-1}\,\mathbf{Q}_i^t)$. Group sensitivity factor, $c_g = \sum_{\mathbf{A}_i\in g}\Delta^2_{\mathbf{A}_i}$. The distribution of δ among groups satisfies the condition that for any two groups $g,\ g'\in I$, $\frac{\sqrt[3]{b_g c_g}}{\delta_g \ln^{2/3}(2/\delta_g)} = \frac{\sqrt[3]{b_{g'} c_{g'}}}{\delta_{g'} \ln^{2/3}(2/\delta_{g'})}$, from which we can solve δ_g for each group. Then the distribution of ϵ is therefore $\epsilon_g = \epsilon\sqrt[3]{\ln(2/\delta_g)b_g c_g}/Z$, where $Z = \sum_{g\in I}\sqrt[3]{\ln(2/\delta_g)b_g c_g}$.

Example 4.8. Let a model $\mathcal{Q}$ with three matrices be partitioned into two groups as $\{(\mathbf{Q}_1,\mathbf{Q}_2),(\mathbf{Q}_3)\}$. Suppose the trace values $\mathbf{b} = [1, 10, 1000]$ and sensitivity of strategies are all equal to 1. Under $(1, 0.01)$-matrix mechanism, the distribution of ϵ and δ $\{0.23, 0.77\}$ and $\{0.002, 0.008\}$, gives us the minimum error of this partition. This means, we run matrix mechanism on $\mathbf{Q}_1$ and $\mathbf{Q}_2$ each with privacy budget $(0.23, 0.002)$ and $\mathbf{Q}_3$ with budget $(0.77, 0.008)$.

4.5.3.4 Choosing an optimal partition

The next step is to choose a partition I that minimizes $\textsc{MinErr}_{\mathcal{M}}(I)$ over all valid partitions $\mathcal{I}$. As the total number of partitions is exponential in n, a naive search algorithm will cost exponential time to find the optimal partition. We propose a heuristic algorithm limiting searching a polynomial space based on the following observation.

Consider a model with only two query matrices. It is easy to find out that parallel composition is better and reaches the most advantage when $b_1/c_1 = b_2/c_2$, where group trace value b_i and group sensitivity factor c_i are defined in Equation (4.5).

$\mathbf{x}_1$	cell condition
0	C.age≤40
3	C.age>40

(a) $\mathbf{x}_\mathsf{C}$

$\mathbf{x}_2$	cell condition
9	O.year=2010,C.age≤40
2	O.year=2010,C.age>40
7	O.year=2011,C.age≤40
6	O.year=2011,C.age>40

(b) $\mathbf{x}_\mathsf{O}$

Figure 4.7: Non-realizable data vectors as $\mathbf{x}_\mathsf{C}$ indicates no customer is younger than 40 while $\mathbf{x}_\mathsf{O}$ shows there must be some.

Assume ϕ is the angle between vectors (b_1, c_1) and (b_2, c_2) in a 2-dimensional space, this means $\phi = 0$. Sequential composition only benefits when ϕ is large. When coming to n-sized model, we can apply the similar idea: we keep elements in each group close to each other (smaller ϕ) and large difference across groups (bigger ϕ).

We say partition I of a set is a *refinement* of a partition I' of the same set, if every element of I is a subset of some element of I', noted as $I \preceq I'$. This means elements in I' can be obtained by combine some elements in I. In such cases, we say I is *finer* than I' and I' is *coarser* than I. E.g., consider the fully split partition I_s and fully joined partition I_j, we have $I_s \preceq I_j$. In fact, $(\mathcal{I}, \preceq)$ defines a complete lattice. We use $csr(I)$ to denote all partitions that one-step coarser than I, meaning each of which is generated by merging exactly two groups in I. The algorithm is to start from I_s, and search the space of $csr(I)$ for the current best partition I at each step and stops when all partitions in $csr(I)$ are worse than I. This procedure reduces the exponential search space to $O(n^3)$ where n is the number of query matrices and our simulation shows it always approaches the optimal partition.

4.6 Sampling synthetic databases

The private, noisy answers to the model queries, generated using the techniques of the last section, are not sufficient for carrying out performance evaluation. It remains to generate a complete synthetic database instance from the perturbed model. The

major challenge results from the fact that a model with perturbed data vectors might not be *realizable*: it is possible that there is no database instance that conforms with the perturbed model statistics. An example is illustrated in Figure 4.7. The idea of consistent data vectors discussed in Section 4.5.2 is only a necessary condition for realizability. Realizability depends on a proper relationship across different data vectors. Unfortunately, existing sampling techniques are designed only for unperturbed, realizable models.

To address this challenge, we propose a two-step approach: first we calculate a realizable model and then sample from it using standard methods proposed from literature (e.g. [2]). Note that these steps use the private perturbed model as input and make no further use of the original database. As a result, there is no impact on the privacy guarantee.

Realizable model

A perturbed model may fail to be realizable largely because the perturbation process does not respect key-foreign key relationships. Intuitively, when you sample from a realizable model, each cell in any data vector should have sufficiently high counts to allow propagation to each of its direct descendants.

Formally, let $\mathbf{x}[\psi]$ denote the summation of the cell counts in a data vector $\mathbf{x}$ that satisfy the condition ψ. For example, in Figure 4.7, $\mathbf{x}_{\mathsf{O}}[\mathsf{O}.\text{year}=2011]=7+6=13$. Define $C_r^s = \dim(\mathbf{x}_r) \cap \dim(\mathbf{x}_s)$, the set of common dimensions between two data vectors $\mathbf{x}_s$ and $\mathbf{x}_r$. We also use E_r^s to represent the dimensions that belong to $\dim(\mathbf{x}_s) - \dim(\mathbf{x}_r)$ and at the same time are attributes of table r or r's ancestors. In Figure 4.7, $C_{\mathsf{C}}^{\mathsf{O}} = \{\mathsf{C}.\text{age}\}$. If $\dim(\mathbf{x}_{\mathsf{O}})$ also includes N.nation, $E_{\mathsf{C}}^{\mathsf{O}} = \{\mathsf{N}.\text{nation}\}$ because the dimension N.nation is not in $\mathbf{x}_{\mathsf{C}}$ and is an attribute of N, an ancestor of C.

Theorem 4.3. *Assume R and S are any two tables such that $S \in desc(R)$ and let $\mathbf{x}_r$ and $\mathbf{x}_s$ be their corresponding data vectors. C_r^s and E_r^s are defined as above. A model is realizable if: $\forall c \in dom(C_r^s)$,*

$$\mathbf{x}_r[C_r^s = c] \geq \sum_{e \in dom(E_r^s)} \lceil \frac{1}{\mathcal{K}(R,S)} \cdot \mathbf{x}_s[C_r^s = c, E_r^s = e] \rceil$$

In Figure 4.7, the violation happens because

$$\mathbf{x}_{\mathsf{C}}[\mathsf{C}.\text{age} \leq 40] \not\geq \lceil \frac{1}{\mathcal{K}(\mathsf{C},\mathsf{O})} \cdot \mathbf{x}_{\mathsf{O}}[\mathsf{C}.\text{age} \leq 40] \rceil$$

In the theorem above, we use propagation constraints defined in Section 4.4.1 to restrict the propagation behavior. In the context of privacy, the information of $\mathcal{K}$ is possibly treated as sensitive information of the original dataset and the data owner could choose not to disclose it to the third party. So from their perspective, they are going to later sample synthetic databases with the assumption that $\mathcal{K}$ is infinity. Or the owner could also release the perturbed version of $\mathcal{K}$.

To calibrate the data vectors and make it realizable, we should make changes to cell counts if necessary. An inference process that minimizes the L_2 distance of all cell counts can then be represented as a quadratic program with least squares as the objective function. However, in real applications, data vectors could be high dimensional with millions of entries, in which case, standard quadratic programming inference could be quite expensive. We design a linear-time approximation which works quite well in our application (See Section 4.7.3). The idea is that whenever the inequality in Theorem 4.3 is violated, we choose to increase minimally the cell counts on the left side of the inequality. To calibrate all data vectors into a realizable state, we test each pair of data vectors in reverse topological order of the schema graph.

4.7 Evaluation

In this section, we implement the modeling and sampling methods proposed in the previous sections and evaluate the accuracy of performance evaluation on synthetic data. We build various models for a given workload, perturb the models, sample a set of synthetic databases, and finally run the original workload on both the original and synthetic databases. The primary goal of the experiments is to compare the accuracy, w.r.t. performance evaluation of the workload, of the non-private and private synthetic databases.

4.7.1 Experimental setup

Datasets and workload We use two datasets conforming to the TPC-H schema, the uniform TPC-H generator[1] with scale factor 1 and the skewed TPC-H (denoted sTPC-H) generator [15], which generates non-uniform columns distributions from a Zipfian distribution, where the Zipf value (z) is set to 1.25. The workload queries are generated from TPC-H query blueprints 1, 3, 6 and 10 with various parameters substitution, which are queries involving up to 4-way joins on primary keys and foreign keys.

Neighboring databases definition We assume the customer table C is the protected entity. For this schema, we only need to constrain the propagation to C's descendants, $\mathcal{K}(\mathsf{C}, \mathsf{O})$ and $\mathcal{K}(\mathsf{O}, \mathsf{L})$. In both datasets, propagation from O to L is uniformly distributed from 1 to 7. By definition of our neighboring databases, a single counting query on Lineitem of TPC-H then has sensitivity $41 * 7 = 287$. It is obvious that the strongest privacy guarantee is offered when the propagation constraint $\mathcal{K}$ is set as large as the maximum. Since the maximum propagation between C and O in sTPC-H is 15935, the sensitivity of the same query is $15935 * 7$. This is indeed

[1] http://www.tpc.org/tpch/

the unavoidable case when conservative propagation constraints will be too big to maintain a reasonable level of perturbation. Thus using the modified $\mathcal{K}$ is one way to avoid bad utility while still providing strong protection to the vast majority of participants in the database. In addition, we also want to show a fair comparison between TPC-H and sTPC-H , so $\mathcal{K}(\mathsf{C}, \mathsf{O})$ is set to 41 and $\mathcal{K}(\mathsf{O}, \mathsf{L})$ is set to 7 for both datasets. In sTPC-H, 99.764% of customers have 41 orders or less, so setting $\mathcal{K}$ to 41 means 99.764% of customers have full protection.

We set $\epsilon = 1, 0.1$ and $\delta = 0.01$. According to Equation (4.2), changing δ from 0.01 to 0.001 has a factor of 1.43. Thus, they are equivalent to $\epsilon = 1.19, 0.119$ and $\delta = 0.001$.

4.7.2 Modeling

We implement the model family described in Section 4.3.2. The null model (NM) serves as a baseline approach because it does not depend on the workload. Table 4.1 shows details about the models. For example, we see an enormous jump in data vectors' size for more complex models. Our algorithm has three phases: selecting a strategy, distributing the privacy budget and adding noise. We want to emphasize that the cost for strategy selection is incurred *only once* for each workload of queries, independent of a particular database or setting of epsilon. Once this cost is incurred, generating perturbed data for any database instance or setting of the privacy parameters is efficient, and we therefore consider the overall cost of the algorithm acceptable. For example, in our experiments, even though we sample synthetic databases based on both datasets, and run experiments under multiple choices of ϵ, we only run the strategy selection step once for C3TM. Later steps of distributing the privacy budget and performing actual perturbation run in approximately 20 seconds, even for C3TM with 10^6 cells in its data vectors. Figure 4.8 tells that our budget allocation algorithm can reach much lower total error than simply splitting budget evenly.

4.7.3 Sampling

The sampling process involves two steps: realization and sampling. We apply the approximated realization introduced in Section 4.6, avoiding expensive quadric programming. In the last row of Table 4.1 we show the L_1 distance on data vectors before and after realization, which is basically negligible compared to the size of database. The running time of realization is less than a couple of seconds for all models.

	NM	C1TM	C2TM	C3TM
# model queries	8	250	338	463
size of data vectors	10^1	10^3	10^5	10^6
modeling time (sec)	5	262	760	3009
changes after realization	1	17	45	60

Table 4.1: Detailed information about models

4.7.4 Utility

To assess the utility of the framework, we run workload queries with synthetic databases and measure the performance metrics of by comparing them with execution using the original databases. We use PostgreSQL, and observe two measures: 1) Estimated cost. The optimizer uses statistics of databases to decide a best query plan, and estimates the running time. 2) Running time, which is actual execution time. Note that these two metrics are not necessarily correlated even in modern DBMS.

Model error In the absence of privacy, we apply standard sampling to generate synthetic databases from unperturbed models and run evaluation tasks on them. The error between the collected measurements from these synthetic instances and the true measurements from the original databases is called model error, which helps us to understand the quality of the selected models. We measure the model error of each metric P by its relative error. Let P_o and P_s be the value of P on the original database

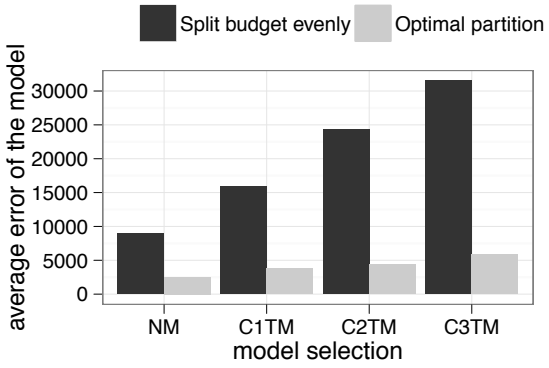


Figure 4.8: Comparison of the error rates for different choices of privacy budget distribution

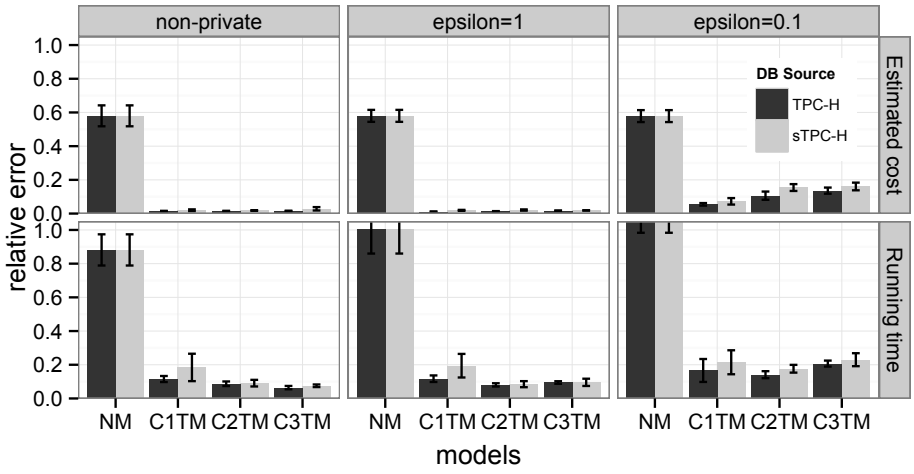


Figure 4.9: Model Error and perturbation error of TPC-H and sTPC-H

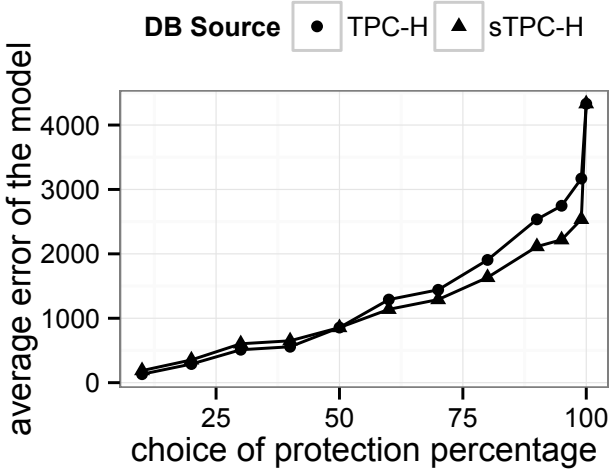


Figure 4.10: Influence of protection percentage on error rates of model C2TM

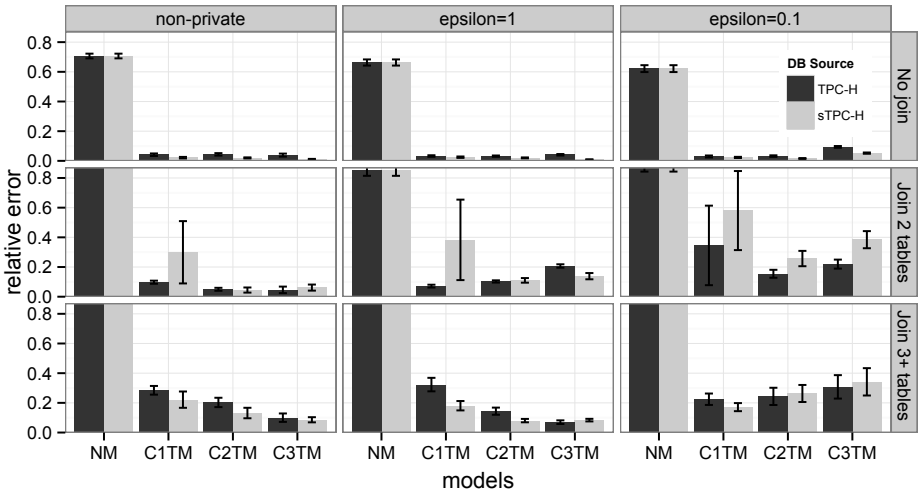


Figure 4.11: Detailed model error and perturbation error of workload running time based on query complexity

and the synthetic database respectively. The relative error of P is $r(P) = \frac{|P_s - P_o|}{P_o}$. The results are summarized in the first column of Figure 4.9, "non-private", where each bar and its error bar represents "mean $\pm$ standard error" of the relative error. We find that all models outperform the baseline model NM significantly, illustrating that the models are effective and that customizing the model to the workload is important. As expected, high-level models (C3TM and C2TM) are better than low-level ones.

Perturbation error Next we want to evaluate the accuracy of performance metrics for databases synthesized using the perturbed private models. We refer to this as perturbation error, which includes model error and the additional distortion of the privacy mechanism. The results, shown in Figure 4.9, are presented in terms of ϵ in the second and third column. With $\epsilon = 1$, our models can handle noise easily, maintaining very small lost of performance compared to non-private case. With $\epsilon = 0.1$, noise becomes more influential. We see obvious increase of both metrics across all models. Because the estimated cost reflects how query planner sees the table statistics and the simpler models add less distortion to model queries, they suffer less from low budget. For actual running time, we see only C2TM can stay in sub-20% for both metrics at $\epsilon = 0.1$.

Utility breakdown Our model series are constructed by stacking up more cross table correlations, so we'd like to differentiate the performance with joins, i.e. no-join, joining two tables and joining 3 or more tables. The breakdown on running time metric is illustrated in Figure 4.11 (We don't show results for estimated cost, because error is much smaller there in all cases.) Note that, in the non-private case, C1TM matches only non-join queries, C2TM can do up to 2-way joins, and C3TM works for up to 3-way joins, all of which are demonstrated in the first column. It turns out that C1TM is not capable of dealing with high joins, especially for the skewed database source. At $\epsilon = 1$, both C2TM and C3TM shows modest distortion, consistent with their overall performance from Figure 4.9. At $\epsilon = 0.1$, C2TM is marginally better than C3TM, across all queries. This is because C3TM's noise level at $\epsilon = 0.1$ finally ruins the benefit of having more information. Overall, C2TM and C3TM are good for $\epsilon = 1$ and C2TM works best for $\epsilon = 0.1$ but only marginally better than C3TM.

Better utility by changing protection percentage At the beginning of this section, we discuss that the strongest privacy is guaranteed when setting propagation

to its maximum value, where 100% of customers is fully protected. However, large propagation increases the sensitivity and thus noise. If reducing the percentage of customers being protected is allowed, we get better utility. E.g., in Figure 4.10, we can almost achieve half perturbation error for C2TM when only fully protecting 90% of customers from sTPC-H .

Outside workload The model-and-sample approach is tailored for particular workloads. The benefit of having synthetic databases is to allow users to run those queries the way they want without privacy budget concerns, e.g., using different DBMS systems. Besides well-modeled workload queries, it is generally interesting to see the performance of outside queries. However, arbitrary queries might not work correctly. E.g., if workload queries do not touch customer's age, none of models will have information for that. Querying customer's age is nothing more than getting randomly generated numbers between 1 and 100. To test outside queries, we randomly combines modeled attributes from any models into multi-joins. Given the size of models and data vectors, this still represents a big space of outside queries. We generate 20 queries, ranging from no joins to 4-way joins, and repeat the evaluation process above. The performance of no-join and 2-way join queries can match up the utility of given workload, if not worse, with all privacy budgets. For queries containing more than three joins, the result becomes unpredictable. However, it is mostly model error that damages the utility, and we do not see much extra distortion from perturbation error.

4.8 Related Work

There have been many proposed methods for synthesizing relational databases [2, 7, 8, 13, 38, 48, 65, 106]. Privacy is a commonly-cited motivation [2, 106]. Yet only one paper actually specifies a privacy condition for generated data [113] and that condition is based on anonymization approaches that lack rigorous gaurantees and may be

vulnerable to a range of attacks demonstrated by the anonymization community. In addition, they do not provide a detailed evaluation of utility, so a comparison with our proposal is difficult.

Among the many works on database synthesis (without privacy), the classical method is to sample databases to derive the data distribution and underlying attribute correlations. Synthesis of the database is then *workload-independent* [13, 38, 48, 106] (i.e., intended to support any set of queries considered) or *workload-aware* [2,8,65] (tailored to a specific workload of interest). We argue that the workload-aware approach is better for database synthesis, since workload-independent approaches may maintain irrelevant information for particular applications, as Seltzer et al. [95] observed. From the perspective of differential privacy, supporting arbitrary workloads requires more noise and results in lower utility. Our modeling method, based on counting queries extracted from workloads, is carefully adapted to the given workload. Many researchers [8, 13] have used cardinality statistics for (non-privately) synthesizing a database. In addition, the idea of building a probabilistic model for a database [34] can potentially improve the accuracy of query estimation.

Differential privacy [25] has been one of the most popular privacy definitions in recent years. Generating differentially private synthetic datasets has been a common goal, but only for single-table schemas [46, 114, 115, 117]. Existing results show that in order to achieve accurate results, the data must be targeted to a constrained set of workload queries. Recent work from Li et al. [63,64] proposed matrix mechanism that is able to compute optimal noise on a set of correlated queries and we extend that to work in a multi-relation setting. The techniques in this chapter are a significant extension to preliminary work [41]. The PINQ framework [74] discusses privacy for multi-relation schemas. However, the protected entity and neighboring databases are not clearly defined and the semantics of the join operation is modified. Lastly, Rastogi

et al [86] consider queries with joins and show that certain limiting assumptions about the adversary can result in improved utility under a model of adversarial privacy.

CHAPTER 5

SHARING PRIVATE EXPONENTIAL RANDOM GRAPH ESTIMATION

This chapter considers how to share estimated parameters of exponential random graph model (ERGM [71]) under differential privacy. As adding direct noise to those parameters of ERGM is not practical, we suggest a two-step solution. Firstly, the perturbation algorithm transforms the sufficient statistics of given ERGM into private version. Secondly, a specially designed parameter estimation process calculates the best parameter using private statistics.

5.1 Introduction

The explosion in the collection of networked data has fueled researchers' interest in modeling networks and predicting their behavior. However, for important application areas such as disease transmission, network vulnerability assessment, and fraud detection (among others), networks contain sensitive information about individuals and their relationships. It is difficult for institutions to release network data and it remains difficult for researchers to acquire data in many important application domains.

Recently, a rigorous privacy standard, *differential privacy* [25] was proposed that allows for formal bounds on the disclosure about individuals that may result from computations on sensitive data. Differential privacy provides each participant in a dataset with a strong guarantee and makes no assumptions about the prior knowledge of attackers.

Since its introduction, differentially private algorithms have been developed for a wide range of data mining and analysis tasks, for both tabular data and networked data. For networks, existing work has focused on algorithms for accurately releasing common graph statistics under differential privacy [45, 55, 79, 83, 90, 109]. However, graph statistics are only one aspect of social network analysis and are often most useful in conjunction with some paradigm for modeling structural features of graphs. Privately modeling graph data has only rarely been explored by researchers; we are aware only of work using the Kronecker model [61] under differential privacy [75].

In this work, we study the differentially private use of the classic exponential random graph model (ERGM) [71, 88, 100]. ERGMs are a powerful statistical modeling tool that allows analysts to analyze a network's social structure and formation process. In social science and related fields ERGMs have been successfully applied to many scenarios, such as co-sponsorship networks [20], friendship networks [36], and corporate and inter-organizational networks [71].

Our goal is to accurately support parameter estimation for ERGMs under differential privacy, focusing on a specific set of model parameters of recent interest to researchers: the *alternating statistics*. These sophisticated statistics represent more structural information than traditional star and triangle counts, and have been shown to lead to much better modeling results [36, 49, 88, 100].

Our adaptation of differential privacy to graphs protects relationships of individuals by limiting the influence on the output of any single relationship (edge) that is created or removed from the network.[1] A standard algorithm that implements this idea is the Laplace mechanism [25], which adds random noise to the output. The amount of noise required is related to the maximum difference in the output due to a

[1]This is one of the most common interpretations of differential privacy for graphs, called *edge* differential privacy [45]. *Node* differential privacy is stronger, but often hurts utility. Our results for edge-differential privacy can easily be extended to k-edge privacy to protect multiple edges.

single edge addition or removal for *any* possible network (this is the *global sensitivity* of the function producing the output). For ERGM estimation, this requires calculating the exact change in the ERGM parameter estimates as a result of changing an edge. Unfortunately, the global sensitivity for most ERGM parameters is either hard to compute in general, or too high, so that using noise calibrated to the global sensitivity is not acceptable.

To overcome this obstacle, we decompose private ERGM estimation into two separate steps. We first privately compute the sufficient statistics for ERGM estimation (typically the model statistics required by model description) and then estimate the parameters using only these sufficient statistics. Since the estimation process uses only the differentially-private statistics, and there is no additional access to the original graph, the output of estimation is also differentially private. In practice, the estimation algorithm is executed either on the server side (by the data owner) or on the client side (by the analyst). In either case, it does not violate the privacy condition to release both the statistics and the derived ERGM parameters.

Challenges arise in both steps of our approach. While prior work has proposed mechanisms for various graph statistics, common ERGM models use unique statistics, e.g., alternating graph statistics [100], which are a complex aggregation of a series of basic graph statistics. We describe new approaches for privately computing these statistics. The second parameter estimation step could be implemented using standard methods [14, 99] while treating the privately-computed statistics as if they were the true statistics. Instead, we propose a novel parameter estimation method based on Bayesian inference, which considers the noise distribution from which the private statistics are drawn and produces more accurate parameter estimates.

Contributions

• In Section 5.3, we describe (ϵ, δ)-differentially private algorithms for estimating two key statistics: alternating k-triangle and alternating k-twopath. The algorithms add

noise proportional to a high-likelihood bound on the local sensitivity of the statistics. Unlike global sensitivity, local sensitivity is determined by the current graph instead of worst-case graphs and can be much lower. Our algorithms use a technique formalized in [55] and inspired by the Propose-Test-Release approach [24].

- We describe a new Bayesian method for ERGM parameter estimation (in Section 5.4) that is designed for the noisy sufficient statistics produced by a differentially private algorithm. While it is possible to use a standard algorithm for estimation, our inference takes the unknown network as a hidden variable and can result in estimates with lower error.
- We study a set of ERGM models based on model terms consisting of alternating graph statistics [100] (in Section 5.5). Our experiments on both synthetic and real graphs show that our techniques significantly reduce noise over baseline approaches.

5.2 Background

5.2.1 Exponential random graph model (ERGM)

A graph $G = (V, E)$ is defined as a set of nodes V and relationships $E : V \times V \rightarrow \{0, 1\}$. A common representation of a graph is as an adjacency matrix x, where $x_{ij} \in \{0, 1\}$ indicating whether there is an edge from node i to j. Let $f(\cdot)$ define a vector of graph statistics called the *model terms*; the concrete values of $f(x)$ are the *model statistics*. Formally, the ERGM with parameter vector θ defines a probability distribution over graphs in the space $\mathcal{X}$ (typically the set of all simple graphs with n vertices):

$$p(x|\theta) = \frac{\exp(\theta \cdot f(x))}{Z_\theta} \tag{5.1}$$

Z_θ is a normalizing constant to make $p(x)$ a true probability distribution, parameterized by θ. If x_0 is the observed graph and X represents the random variable

defined by the distribution above, our goal is to tune the parameter vector θ, s.t. the expected value of $f(X)$ is equal to observed statistics, meaning $\mathbb{E}_\theta(f(X)) = f(x_0)$, which intuitively puts the observed graph in the "center" of space of possible graphs implied by the model. For example, the simplest ERGM uses the number of edges as the only model term. If m_0 is the total number of edges in x_0, the θ, which enables the expected number of edges of ERGM equal to m_0, is given by [78]:

$$\theta = \log \frac{m_0}{\binom{n}{2} - m_0} \tag{5.2}$$

Estimating θ The optimal θ maximizes the likelihood of x_0 given θ [78], i.e., $\arg\max_\theta p(x_0|\theta)$. Unfortunately, most ERGMs do not have an analytical or closed-form estimate for the optimal θ. Thus, numerical solutions are proposed in the literature, such as Markov chain monte carlo maximum likelihood estimation [99] and Bayesian inference [14]. An interesting property of these inference methods is that the algorithm does not require access to the input graph itself, i.e., the sufficient statistics for the parameter estimation are just the model statistics. This feature enables us to decompose the private inference problem into two steps, allowing analysts to see only the sufficient statistics.

Alternating statistics A model term is usually a counting query of a specific graph pattern. Common patterns include triangles, stars and loops [71]. Recent research has introduced alternating statistics for k-star, k-triangle and k-twopath, which can represent structural properties of a graph better than traditional star and triangle counts [100]. Many works have explored these statistics since they were proposed, and they are an active and promising form of ERGM [36, 49, 88, 100]. Our work is focused on these alternating statistics (defined precisely in Section 5.3) which have not been studied before under differential privacy. A wide variety of other model

terms are used with ERGMs; our general approach is compatible with other terms but they are beyond the scope of this work.

5.2.2 Differential privacy

In Chapter 2, we introduced differential privacy and Laplace mechanism. In this chapter, our database is a graph describing relationships among individuals. Our purpose is to protect relationships among individuals so we adapt differential privacy, following [45, 55, 83, 90, 109], by defining a neighboring graph as a graph that differs by one edge.

Local sensitivity and its smooth bound

Recall that differential privacy can be achieved by adding Laplace noise to the output of algorithms according to privacy parameters and query's global sensitivity. The global sensitivity of a query is the maximum possible difference in the output when evaluating the query on two neighboring graphs.

Some common graph analyses have high global sensitivity, requiring the Laplace mechanism to add enormous amounts of noise. For example, consider the simplest ERGM model above where θ is calculated by (5.2). On a graph where $m_0 = 0$ or a graph where $m_0 = \binom{n}{2}$, θ can change drastically with the addition or deletion of one edge. In other words, the global sensitivity is very high for this function. But the fact is that most real graphs are nothing like these extremes. Thus, by only focusing on the input graph's neighbors, the *local sensitivity* [79] can be much smaller.

Definition 5.1 (Local sensitivity [79])**.** Given query f and graph x, local sensitivity $\mathsf{LS}_f(x)$

$$\mathsf{LS}_f(x) = \max_{x,x' neighbors} |f(x) - f(x')|$$

However, one cannot achieve differential privacy by adding noise proportional to the local sensitivity because local sensitivity itself could disclose information. The

authors of [79] proposed using a smooth upper bound on the local sensitivity, the *smooth sensitivity*. Intuitively, smooth sensitivity tries to "smooth" out the difference between local sensitivities of two neighbors, so that it is itself not sensitive. Let $d(x, x')$ be the distance between two graphs, i.e. the number of edges in which they differ:

Definition 5.2 (Smooth bound and smooth sensitivity [79])**.** Function $\mathsf{S}_f : \mathcal{X} \Rightarrow R$ defines a β-smooth bound of local sensitivity on query f if

$$\begin{aligned} \forall x \ &: \ \mathsf{S}_f(x) \geq \mathsf{LS}_f(x) \\ \forall x, x' \text{ neighbors } &: \ \mathsf{S}_f(x) \leq \exp(\beta)\mathsf{S}_f(x') \end{aligned}$$

The β-smooth sensitivity of f is a β-smooth bound, and

$$\mathsf{SS}_{f,\beta}(x) = \max_{x'} \{\mathsf{LS}_f(x') \cdot \exp\left(-\beta d(x, x')\right)\}$$

Calculating the smooth sensitivity for a function may be easy (in cases like the median of a list of numbers [79]) but could be quite difficult for other functions, requiring complex proofs and nontrivial algorithms [55]. Even though smooth sensitivity may provide tight bound for local sensitivity, we show that it is NP-hard for two alternating statistics commonly used in ERGMs.

5.3 Perturbing model statistics

In this section we provide methods for privately computing alternating graph statistics. In Sec. 5.3.1 we define three alternating statistics and show that one of them (alternating k-star) has a constant global sensitivity. This means the Laplace mechanism to be applied with relatively small error. However, alternating k-triangle and alternating k-twopath both have high global sensitivity. In Sec 5.3.2 we show that we cannot resort to smooth sensitivity, as calculating the smooth sensitivity

is NP-hard in both cases. To address this challenge, we adapt a technique which calibrates noise to a private, high-likelihood upper bound on the local sensitivity [55]. That bound is produced using the global sensitivity of the local sensitivity function. If, however, the global sensitivity of that function is high, the technique can be repeatedly applied, using a high-likelihood bound on the local sensitivity of the local sensitivity function. We describe these "first-order" and "second-order" algorithms in Sec 5.3.2 and then analyze the local sensitivity of alternating k-triangle and alternating k-twopath in Sec. 5.3.3.

5.3.1 Alternating graph statistics

The three alternating graph statistics, alternating k-star, alternating k-triangle and alternating k-twopath, are essentially complex aggregations of traditional k-star, k-triangle and k-twopath statistics. Instead of considering a vector of k terms, the alternating statistics aggregate over the terms but enforce alternating signs between each consecutive term, to weaken the correlation among different terms and effectively reduce the weight on higher terms near k.

Alternating k-star The k-star is a counting query of a star pattern in the graph, where each star contains k edges, i.e., $S_k = \sum_i \binom{d_i}{k}$ where d_i is the degree of node i.

Definition 5.3 (Alternating k-star [100])**.** With parameter $\lambda \geq 1$, alternating k-star S is defined as

$$S(x;\lambda) = S_2 - \frac{S_3}{\lambda} + \ldots + (-1)^{n-1}\frac{S_{n-1}}{\lambda^{n-3}}$$

The λ parameter here is a good way to control the geometrical weights on all k-stars.

Alternating k-triangle A k-triangle is a graph pattern in which k triangles share a common edge. The k-triangle query asks for the total number of k-triangles in the graph. Define the shared partner matrix C, where each entry (i, j) in C is the

count of shared partners between nodes i and j, mathematically $C_{ij}(x) = \sum_l x_{il}x_{lj}$. Formally, k-triangle T_k is defined:

$$T_k = \sum_{i<j} x_{ij}\binom{C_{ij}}{k} \quad (k \geq 2), \quad \text{and } T_1 = \frac{1}{3}\sum_{i<j} x_{ij}C_{ij}$$

Alternating k-triangle is defined similarly as alternating k-star, using parameter λ:

Definition 5.4 (Alternating k-triangle [100])**.** With parameter $\lambda \geq 1$, alternating k-triangle T is:

$$T(x;\lambda) = 3T_1 - \frac{T_2}{\lambda} + \frac{T_3}{\lambda^2} - \ldots + \left(\frac{-1}{\lambda}\right)^{n-3} T_{n-2}$$

Alternating k-twopath A k-twopath graph pattern is very similar to k-triangle, except it does not require the shared edge required by the k-triangle statistic. Using the shared partners matrix C above, the counting query for k-twopath U_k is:

$$U_k = \sum_{i<j} \binom{C_{ij}}{k} \quad (k \neq 2), \quad \text{and } U_2 = \frac{1}{2}\sum_{i<j} \binom{C_{ij}}{2}$$

And alternating k-twopath is:

Definition 5.5 (Alternating k-twopath [100])**.** With parameter $\lambda \geq 1$, alternating k-twopath U is

$$U(x;\lambda) = U_1 - \frac{2}{\lambda}U_2 + \sum_{k=3}^{n-2}\left(\frac{-1}{\lambda}\right)^{k-1} U_k$$

Alternating k-star S is the only statistic that can be readily solved using existing privacy mechanisms. Because the degree sequence is a sufficient statistic for S, one natural approach is to use the mechanism described by Hay et al [45] to compute a private degree sequence from x, and then use it to compute S by Eq. (5.3). But, in

fact, it can be shown that the global sensitivity of S is at most 2λ. Thus, Laplace noise may be a better choice (λ is usually set to a small integer in practice). We make empirical comparisons between these methods in Section 5.5.

Lemma 5.1. *The global sensitivity of alternating k-star is at most 2λ.*

5.3.2 Bounding local sensitivity

Because the global sensitivity of alternating k-triangle and k-twopath can be as large as $O(n)$, we would like to use a method which adds noise scaled to the local sensitivity or a quantity close to the local sensitivity. One approach is to compute a smooth bound on the local sensitivity, however, the following lemma shows the NP-hardness of computing such a bound for these two statistics:

Lemma 5.2. *Computing the smooth sensitivity for both alternating k-triangle and alternating k-twopath is NP-hard.*

We therefore employ a technique inspired by the Propose-Test-Release framework [24], and formalized by Karwa et al [55], where it was used to estimate k-triangles. The technique first computes a private over-estimate of the local sensitivity, one that is higher than the local sensitivity with high probability. That becomes a safe sensitivity value for calibrating Laplace noise, however, the result satisfies only the weaker notion of (ϵ, δ)-differential privacy.

Let $f(x)$ be the sensitive function/query. We use $\mathsf{LS}_{f,1}(x)$ to denote the local sensitivity of f, which is a function of the input graph x.

Algorithm 5.1 Local sensitivity bounding algorithm (First order)

Require: input graph x, query f and ϵ, δ

1: $a = \frac{\ln(1/\delta)}{\epsilon}$
2: $\tilde{y}_1 = \mathsf{LS}_{f,1}(x) + Lap(\mathsf{GS}(\mathsf{LS}_{f,1}(x))/\epsilon) + a \cdot \mathsf{GS}(\mathsf{LS}_{f,1}(x))$
3: $\tilde{y} = f(x) + Lap(\tilde{y}_1/\epsilon)$
4: **return** $\tilde{y}, \tilde{y}_1$

In Algorithm 5.1, $\tilde{y}_1$ is the private bound on the local sensitivity, computed by adding scaled noise to $\mathsf{LS}_{f,1}(x)$, as well a positive offset, so that the bound is higher than $\mathsf{LS}_{f,1}(x)$ with high probability. Notice that the scale of the noise is determined by the global sensitivity of the local sensitivity, $\mathsf{GS}(\mathsf{LS}_{f,1}(x))$.

If $\mathsf{GS}(\mathsf{LS}_{f,1}(x))$ is large, it may cause $\tilde{y}_1$ to be a significant over-estimate of $\mathsf{LS}_{f,1}(x)$. We can repeat this approach by using a safe upper bound of the local sensitivity of $\mathsf{LS}_{f,1}(x)$, as presented below. Thus, Algorithm 5.1 bounds the first-order local sensitivity and the following algorithm bounds the second-order local sensitivity.

Algorithm 5.2 Local sensitivity bounding algorithm (Second order)

Require: input graph x, query f and ϵ, δ

1: $a = \frac{\ln(1/\delta)}{\epsilon}$
2: $\tilde{y}_2 = \mathsf{LS}_{f,2}(x) + Lap(\mathsf{GS}(\mathsf{LS}_{f,2}(x))/\epsilon) + a \cdot \mathsf{GS}(\mathsf{LS}_{f,2}(x))$
3: $\tilde{y}_1 = \mathsf{LS}_{f,1}(x) + Lap(\tilde{y}_2/\epsilon) + a \cdot \tilde{y}_2$
4: $\tilde{y} = f(x) + Lap(\tilde{y}_1/\epsilon)$
5: **return** $\tilde{y}, \tilde{y}_1, \tilde{y}_2$

Theorem 5.3. *Algorithm 5.1 is* $(2\epsilon, \frac{1}{2}e^{\epsilon}\delta)$*-differential privacy. Algorithm 5.2 is* $(3\epsilon, \frac{1}{2}e^{\epsilon}\delta + \frac{1}{2}e^{2\epsilon}\delta)$*-differential privacy.*

The proof of Theorem 5.3 relies on the Lemma 4.4 from [55], as we restate it as follows:

Lemma 5.4. *If* M *is* (ϵ_1, δ_1)*-differentially private, and* $\Pr[M(x) \geq \mathsf{LS}_f(x)] > 1 - \delta_2$ *for all* x*, the following* A *which returns a pair of values,*

$$A(x) = (\, M(x), Lap(M(x)/\epsilon_2)\,)$$

is $(\epsilon_1 + \epsilon_2, \delta_1 + e^{\epsilon_1}\delta_2)$*-differentially private.*

Proof of Theorem 5.3. In Algorithm 5.1, $\tilde{y}_1$ is ϵ-differentially private as it is based on Laplace mechanism and post-processing (adding positive offset). Let $g_1 = \mathsf{GS}(\mathsf{LS}_{f,1}(x))$. If the sampled Laplace noise in line 2 is b,

$$\begin{aligned}\Pr[\tilde{y}_1 \leq \mathsf{LS}_{f,1}(x)] &= \Pr[b \leq -ag_1] \\ &= \int_{-\infty}^{-ag_1} \frac{1}{2g_1/\epsilon} exp(-|x|\epsilon/g_1)\,\mathrm{d}x \\ &= \frac{1}{2} exp(-ag_1 * \epsilon/g_1) = \frac{\delta}{2}\end{aligned}$$

So, $\Pr[\tilde{y}_1 \geq \mathsf{LS}_{f,1}(x)] > 1 - \delta/2$. By applying Lemma 5.4, the Algorithm 5.1 is $(\epsilon + \epsilon, 0 + e^{\epsilon}\frac{\delta}{2})$-differential privacy, which is $(2\epsilon, \frac{1}{2}e^{\epsilon}\delta)$-differential privacy.

In Algorithm 5.2, both $\tilde{y}_1$ and $\tilde{y}_2$ are ϵ-differentially private. Similarly, $\Pr[\tilde{y}_2 \geq \mathsf{LS}_{f,2}(x)] > 1 - \delta/2$. As above, by applying Lemma 5.4, $\tilde{y}_1$ is $(2\epsilon, \frac{1}{2}e^{\epsilon}\delta)$-differential privacy. Furthermore, applying Lemma 5.4 one more time, the final $\tilde{y}$ is $(2\epsilon+\epsilon, \frac{1}{2}e^{\epsilon}\delta + e^{2\epsilon}\frac{\delta}{2})$-differential privacy, which is $(3\epsilon, \frac{1}{2}e^{\epsilon}\delta + \frac{1}{2}e^{2\epsilon}\delta)$-differential privacy.

□

The step of replacing the global sensitivity by a high-likelihood bound on the local sensitivity can be repeatedly applied to form more complex higher order algorithms. However, each additional bounding step requires splitting the privacy budget and the combined effects of repeatedly over-estimating higher order sensitivities may diminish utility. For the two alternating statistics we consider, and the datasets we tested on, we found that first order and second order is sufficient.

Error analysis

Definition 5.6. $Y_0, Y_1, \ldots, Y_n$ is a *random variable chain*, when the following condition is satisfied: for any $i \in [0, n-2]$, Y_i is conditionally independent of $Y_{i+2}, Y_{i+3}, \ldots Y_n$ given Y_{i+1}.

From conditional independence, an important property of random variable chain is the following:

$$\Pr(Y_i|Y_{i+1}, Y_{i+2}, \ldots, Y_n) = \Pr(Y_i|Y_{i+1})$$

It is easy to see that $\tilde{y}, \tilde{y}_1, \ldots$ is actually a random variable chain. We use mean squared error (MSE) as the measurement of error. In Algorithm 5.1 and 5.2, MSE of $\tilde{y}$ can be written as $\mathbb{E}[(\tilde{y} - f(x))^2] = \mathbb{V}[\tilde{y}] + (\mathbb{E}[\tilde{y}] - f(x))^2$. Since $\tilde{y}$ is always unbiased (Laplace noise in the last step with mean zero), MSE $= \mathbb{V}[\tilde{y}]$.

Without knowing the true value of the local sensitivities, it is quite hard to compute the MSE. That is to say, we cannot compute the error like we do for the Laplace mechanism, since the noise in the latter is independent of input graph x. But, by exploring properties of the random variable chain, it is possible to utilize the following Lemma as a closed form calculation tool for MSE. In fact, we extend law of total expectation/variance [111].

Lemma 5.5. *$Y_0, Y_1, \ldots, Y_n$ is a random variable chain. Write $\bigsqcup_{n,i} \mathbb{E}[\cdot]$ as a shortcut for $\mathbb{E}_{Y_n}[\mathbb{E}_{Y_{n-1}|Y_n}[\ldots \mathbb{E}_{Y_i|Y_{i+1}}[\cdot]]]$. Then*

$$\begin{aligned}
\mathbb{E}[Y_0] =& \bigsqcup_{n,0} \mathbb{E}[Y_0] \\
\mathbb{V}[Y_0] =& \bigsqcup_{n,1} \mathbb{E}[\underset{Y_0|Y_1}{\mathbb{V}}[Y_0]] \\
&+ \sum_{i=2}^{n-2} \left(\bigsqcup_{n,i} \mathbb{E}[\underset{Y_{i-1}|Y_i}{\mathbb{V}}[\bigsqcup_{i-2,0} \mathbb{E}[Y_0]]] \right) + \underset{Y_n}{\mathbb{V}}[\bigsqcup_{n-1,0} \mathbb{E}[Y_0]]
\end{aligned}$$

Applying Lemma 5.5 , one can calculate MSE of Algorithm 5.1 and 5.2. Such error measurement can serve an evaluation tool for privacy researchers when working with our algorithms. From the perspectives of data owners, the analytic result of MSE can help them to decide between Algorithm 5.1 and 5.2, i.e., with fixed privacy parameters, selecting the algorithm with less error (more utility).

5.3.3 Alternating k-triangle and k-twopath

Now we apply the idea of local sensitivity bounding to alternating k-triangle and alternating k-twopath. Let $\beta = 1 - 1/\lambda$. By binomial coefficients, we can rewrite alternating k-triangle $T(x;\lambda)$ as

$$T(x;\lambda) = \lambda \sum_{i<j} x_{ij} \left\{1 - \beta^{C_{ij}}\right\} \tag{5.3}$$

Lemma 5.6. *Set $C'_{iv} = C_{iv} - x_{ij}$ and $C'_{vj} = C_{vj} - x_{ij}$. Let N_{ij} be all shared partners of node i and j and $C_{max} = \max_{i<j} C_{ij}$. The local sensitivity of T is*

$$\mathsf{LS}_{T,1}(x) = \max_{i<j} \ \lambda \left\{1 - \beta^{C_{ij}}\right\} + \sum_{v \in N_{ij}} \left\{\beta^{C'_{iv}} + \beta^{C'_{vj}}\right\} \tag{5.4}$$

$$\leq \lambda + 2C_{max} \tag{5.5}$$

As C_{max} has global sensitivity 1, $\mathsf{LS}_{T,1}$ has global sensitivity at most 2. So we can construct a first-order local sensitivity bound using $\mathsf{LS}_{T,1} = \lambda + 2C_{max}$ to compute private alternating k-triangle.

For alternating k-twopath $U(x;\lambda)$, we can rewrite it as

$$U(x;\lambda) = \lambda \sum_{i<j} \left\{1 - \beta^{C_{ij}}\right\} \tag{5.6}$$

Lemma 5.7. *Let N_i be the set of neighbors of node i and d_{max} be the maximum degree. Set $C'_{iv} = C_{iv} - x_{ij}$ and $C'_{vj} = C_{vj} - x_{ij}$. We have local sensitivity*

$$\mathsf{LS}_{U,1}(x) = \max_{i<j} \left\{ \sum_{v \in N_i, v \neq j} \beta^{C'_{vj}} + \sum_{v \in N_j, v \neq i} \beta^{C'_{iv}} \right\} \tag{5.7}$$

$$\leq 2d_{max} \tag{5.8}$$

$$\mathsf{LS}_{U,2}(x) \leq \frac{\max(4, 1 + C_{max})}{\lambda} \tag{5.9}$$

From Lemma 5.7 above, (5.8) has global sensitivity 2, since d_{max} will change by at most 1 by adding or removing an edge. (5.9) has global sensitivity $1/\lambda$ for $C_{max} > 3$. Therefore, we can construct either first-order or second-order bound. Note that (5.7) is the exact local sensitivity of alternating k-twopath, but we cannot bound it in Algorithm 5.1 as (5.7)'s global sensitivity is not clear. Instead we use (5.8). When applying Algorithm 5.2, (5.7) is the local sensitivity to be bounded at Line 4, as by that step the higher order (second-order) local sensitivity (5.9) has already be safely bounded. We will compare the resulting error empirically in Section 5.5.

5.4 ERGM parameter estimation

The parameter estimation step in our workflow takes the private sufficient statistics $\tilde{y}$ from the previous perturbation step and finds the best parameter vector θ. As stated above, this step is essentially post-processing a differentially private output, so the output θ is also differentially private. In this section, we discuss different ways of estimating θ given $\tilde{y}$.

5.4.1 Standard estimation

Current estimation techniques [14, 99] provide a baseline solution for parameter estimation with private statistics. As these procedures essentially only need access to model statistics, our sufficient statistics in $\tilde{y}$ take the place of the true model terms. The semantics is now to search for θ that defines a probability distribution on graphs with expected model statistics equal to $\tilde{y}$. Intuitively, the utility of this method

depends on the amount of noise added into y_0 and how θ reacts to those changes in y_0.

Prior to applying standard estimation, we post-process $\tilde{y}$ to cope with some of the difficulties of the perturbed model statistics. As the output of perturbed $\tilde{y}$ might not be *graphical* (i.e., no graph has statistics equal to $\tilde{y}$), standard estimation may fail to converge. We propose generating a graph that has the closest statistics to $\tilde{y}$ and use the statistics from that graph to replace $\tilde{y}$, in order to avoid non-converging situations and to potentially remove noise from $\tilde{y}$ simultaneously. We use simulated annealing for this purpose and, in practice, we often see big improvements in the accuracy of estimates.

5.4.2 Bayesian inference

Standard estimation is the direct way of post-processing $\tilde{y}$, but since we know the distribution of the noise added to $\tilde{y}$, we can "guess" the true values and incorporate them into the estimation algorithm. This idea naturally fits into *Bayesian inference* based post-processing. While based on earlier work [14] on Bayesian inference for non-private estimation, our method deals with the extra hidden variable of graph x in our setting. And later we will see, by introducing the unknown x, our method can utilize more information from private statistics, such as the local sensitivity bounds. In particular, we search for θ given $\tilde{y}$, represented as the posterior distribution of ERGM parameter θ:

$$\begin{aligned} p(\theta|\tilde{y}) \propto p(\tilde{y}|\theta)p(\theta) &= \sum_x p(\tilde{y}|x)p(x|\theta)p(\theta) \\ &= \sum_x p(\tilde{y}|x)q(x;\theta)p(\theta)/Z_\theta \end{aligned} \tag{5.10}$$

where x is our guess about x_0, but the fact is that we need to summarize over all possible x to get to the posterior. In (5.10), $p(\tilde{y}|x)$ is the *privacy distribution*, defined by the differential privacy mechanism applied on sufficient statistics. $p(x|\theta)$ is

the *ERGM distribution*, as shown in (5.1) and $q(x;\theta)$ represents the unnormalized distribution.

$$q(x;\theta) = \exp(\theta \cdot f(x)) \tag{5.11}$$

The probability distribution (5.10) is hard to calculate or even sample from directly due to summarization over all graphs and normalizing constant Z_θ. Using the exchange algorithm [77], we introduce extra variables x, θ' and x' to bypass the difficult terms (5.10). By carefully choosing the probability distribution of these new random variables, the posterior distribution is now augmented as shown in (5.12). The key is that the marginal posterior distribution for θ in (5.12) is equivalent to (5.10). Thus, if we are able to sample from the distribution in (5.12), the marginal posterior distribution for θ can be obtained by summarizing over all samples.

$$p(\theta, x, \theta', x'|\tilde{y}) \propto p(\tilde{y}|x)p(x|\theta)p(\theta)p(\theta'|\theta)p(x'|\theta') \tag{5.12}$$

θ' is sampled from *proposal distribution* $p(\theta'|\theta)$, where, for a given θ, a new θ' can be proposed according to $p(\theta'|\theta)$. A common choice is a multivariate normal distribution or a multivariate t distribution, with mean equal to θ. x, x' are sampled graphs under the ERGM with parameter θ and θ'.

Algorithm 5.3 ERGM parameter estimation with private model statistics

Require: $\tilde{y}$, initial θ, x

1: **for** i in 1 to T **do**
2: Sample $\theta' \sim p(\theta'|\theta)$
3: Sample $x' \sim p(x'|\theta')$
4: Replace θ with θ' and x with x', with probability $\min(1, H)$ //H by (5.13) below
5: **return** samples of θ.

A MCMC based sampling process for (5.12) is shown in Algorithm 5.3. In particular, the initial input θ and x could be any parameters and any graph. In Line 3, we need a separated MCMC chain to sample $x' \sim p(x'|\theta')$. In such MCMC algorithms, at each iteration, we propose adding or removing edges in the current state of graph, calculate the new model statistics, compare the probability of new state x_{new} to that of old state x_{old}, and with probability $p(x_{new}|\theta')/p(x_{old}|\theta')$ the change is accepted. This process should be run long enough so that final sample x' is truly from $p(x'|\theta')$.

H in Line 4 is the ratio of accepting the exchange, computed by comparing the probability before and after exchange. That is, we exchange θ with θ' and x with x' in (5.12) and calculate the ratio. Then the complex terms are cancelled out and each remaining term is easy to compute.

$$\begin{aligned} H &= \frac{p(\tilde{y}|x')p(x'|\theta')p(\theta')p(\theta|\theta')p(x|\theta)}{p(\tilde{y}|x)p(x|\theta)p(\theta)p(\theta'|\theta)p(x'|\theta')} \\ &= \frac{p(\tilde{y}|x')p(\theta')p(\theta|\theta')}{p(\tilde{y}|x)p(\theta)p(\theta'|\theta)} \end{aligned} \tag{5.13}$$

In practice, Algorithm 5.3 usually results in low acceptance rates in the exchange step in Line 4 and thus long mixing times for the MCMC process. We now propose to separate that last step, isolating simultaneously updated θ and x into two different steps, as shown in Algorithm 5.4, which improves the acceptance rate significantly.

Algorithm 5.4 Improved ERGM parameter estimation with private model statistics

Require: $\tilde{y}$, initial θ, x

1: **for** i in 1 to T **do**
2: Sample $\theta' \sim p(\theta'|\theta)$
3: Sample $x' \sim p(x'|\theta')$
4: Exchange θ with θ', with probability $\min(1, H_1)$ $//H_1$ by (5.14) below
5: Replace x with x', with probability $\min(1, H_2)$ $//H_2$ by (5.15) below
6: **return** samples of θ.

H_1 and H_2 in Algorithm 5.4 are defined as follows.

$$
\begin{aligned}
H_1 &= \frac{p(\tilde{y}|x)p(x|\theta')p(\theta')p(\theta|\theta')p(x'|\theta)}{p(\tilde{y}|x)p(x|\theta)p(\theta)p(\theta'|\theta)p(x'|\theta')} \\
&= \frac{q(x;\theta')p(\theta')p(\theta|\theta')q(x';\theta)}{q(x;\theta)p(\theta)p(\theta'|\theta)q(x';\theta')}
\end{aligned} \tag{5.14}
$$

$$
\begin{aligned}
H_2 &= \frac{p(\tilde{y}|x')p(x'|\theta)p(\theta)p(\theta'|\theta)p(x|\theta')}{p(\tilde{y}|x)p(x|\theta)p(\theta)p(\theta'|\theta)p(x'|\theta')} \\
&= \frac{p(\tilde{y}|x')q(x';\theta)q(x;\theta')}{p(\tilde{y}|x)q(x;\theta)q(x';\theta')}
\end{aligned} \tag{5.15}
$$

The correctness of Algorithm 5.4 can be proved briefly in terms of a component-wise Metropolis-Hasting algorithm, with hybrid Gibbs updating steps. In each iteration, θ' and x' (Line 2 and 3) are drawn based on full conditional distribution, so the updating probability is always 1. In Line 4 and 5, we update θ and x with Hasting ratios. Although we may end up updating θ' and x' more times in a iteration, we still get to the detailed balance in MCMC [32].

When applying Algorithm 5.4 to real ERGM models, the key is correctly computing H_1 and H_2. Everything in H_1 is independent of the privacy mechanism used for the model terms. In H_2, the ratio of privacy distribution $\frac{p(\tilde{y}|x')}{p(\tilde{y}|x)}$ is mechanism dependent. Here, we illustrate the cases for both Laplace mechanism and local sensitivity bounding algorithms.

Example 5.7 (Laplace mechanism)**.** If the Laplace mechanism is applied on all model terms (f_i for i-th model term) independently, and $\tilde{y}$, ϵ and GS are the vectors of private statistics, privacy parameters and global sensitivities respectively, $p(\tilde{y}|x)$ is then:

$$
p(\tilde{y}|x) \propto \exp\left(-\sum_i |\tilde{y}_i - f_i(x)|\epsilon_i/\mathsf{GS}_i\right) \tag{5.16}
$$

Assume we use a symmetric proposal distribution for θ, i.e., $p(\theta'|\theta) = p(\theta|\theta')$. With Algorithm 5.4, ratio H_1 and H_2 can be written as (after taking logarithm)

$$\log H_1 = \log \frac{p(\theta')}{p(\theta)} + (\theta - \theta') \cdot (f(x') - f(x)) \tag{5.17}$$

$$\log H_2 = (\theta - \theta') \cdot (f(x') - f(x)) + \sum_i \frac{\epsilon_i}{\mathsf{GS}_i} \left(|\tilde{y}_i - f_i(x)| - |\tilde{y}_i - f_i(x')|\right) \tag{5.18}$$

Example 5.8 (Local sensitivity bounding)**.** Assume a single model term and first-order local sensitivity bounding (multiple model terms and second order can be adjusted accordingly), and privacy parameter ϵ and δ is the input for Algorithm 5.1. Let $a = \ln(1/\delta)/\epsilon$.

In the process of MCMC, for current sampled graph x, we write l_1 as the local sensitivity on x and g_1 as the global sensitivity of local sensitivity. The first-order local sensitivity bounding (Algorithm 5.1) returns $\tilde{y}, \tilde{y}_1$ for the observed graph. $p(\tilde{y}, \tilde{y}_1|x)$ can be represented as follows by omitting terms that will be cancelled out later in calculating $\frac{p(\tilde{y},\tilde{y}_1|x')}{p(\tilde{y},\tilde{y}_1|x)}$.

$$\begin{aligned} p(\tilde{y}, \tilde{y}_1|x) &= p(\tilde{y}|x, \tilde{y}_1)p(\tilde{y}_1|g_1, l_1) \\ &\propto \exp\left(-\frac{|\tilde{y} - f(x)|}{\tilde{y}_1/\epsilon} - \frac{|\tilde{y}_1 - l_1 - ag_1|}{g_1/\epsilon}\right) \end{aligned} \tag{5.19}$$

Calculation of $p(\tilde{y}, \tilde{y}_1|x)$ deals with not only the private version of local sensitivity $\tilde{y}_1$, but also more statistics from the sampled graph in each iteration of MCMC (local sensitivity l_1). Recall in the standard estimation, none of them is incorporated in the process. In the next section, we empirically show that such extra information

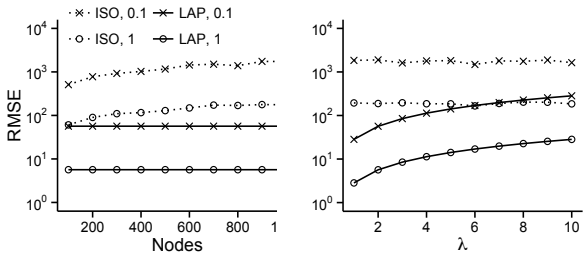


Figure 5.1: Perturbation error on alternating k-star on synthetic graphs. Left: $p = \log(n)/n$, varying size of graph n. Right: $n = 1000$, varying λ.

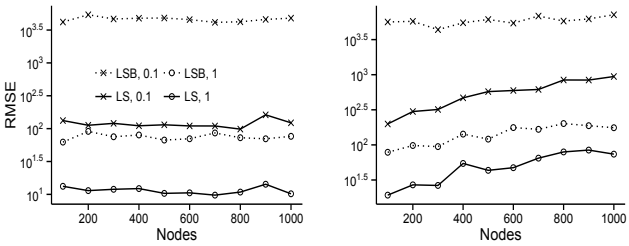


Figure 5.2: Perturbation error for alternating k-triangle. Left: $p = \log(n)/n$. Right: $p = 0.1$. Trend lines for LS are non-private.

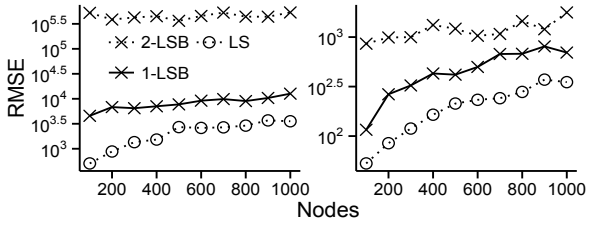
can benefit the estimation. As in the example above, assume a symmetric proposal distribution. With Algorithm 5.4, ratio H_1 is the same as (5.17). H_2 is:

$$\log H_2 = (\theta - \theta') \cdot (f(x') - f(x)) + \frac{|\tilde{y} - f(x)| - |\tilde{y} - f(x')|}{\tilde{y}_1/\epsilon} + \frac{|\tilde{y}_1 - l_1 - ag_1| - |\tilde{y}_1 - l_1' - ag_1|}{g_1/\epsilon} \tag{5.20}$$

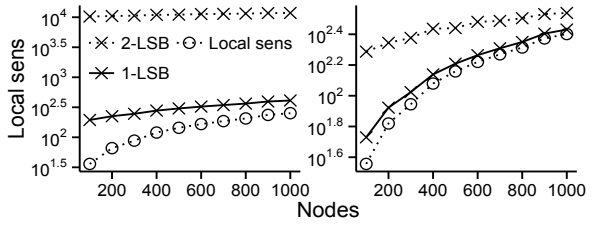
Releasing θ In Algorithm 5.3 and 5.4, we use multiple sampled θ to represent the marginal distribution on θ. A straightforward way to generate a single instance of estimated θ is to calculate the average of those samples. However, in practice, we found that marginal maximum a posterior (MMAP) could give analysts better estimates instead. Formally, MMAP of θ is defined as $\operatorname{argmax}_{\theta} p(\theta|\tilde{y})$. A fast method we apply is reusing the samples of θ from Algorithm 5.4, and performing approximate MMAP estimation by histogram or density estimation. More sophisticated solutions will require further expanding (5.12) before MCMC sampling [22, 54].

5.5 Evaluation

Our evaluation has two goals. First we assess the perturbation error of our privacy mechanisms, particularly the Laplace mechanism on alternating k-star and the local



(1) $p = 0.1, \epsilon = 0.1$ (2) $p = 0.1, \epsilon = 1$



(3) $p = 0.1, \epsilon = 0.1$ (4) $p = 0.1, \epsilon = 1$

Figure 5.3: Perturbation error (1), (2) and local sensitivity values (3), (4) for alternating k-twopath.

sensitivity bounding algorithms on alternating k-triangle and k-twopath. Second, we evaluate the ERGM parameter estimation with private statistics using different approaches proposed in Section 5.4. All our experiments are run on Linux servers with Intel Xeon CPU and 8GB memory. In the experiments, we differ privacy parameters ϵ and δ. Note that, whenever we clarify a value for ϵ or δ, it always means the *overall* budget of the entire perturbation process.

5.5.1 Perturbation error

Our datasets include synthetic and real graphs. Synthetic graphs are generated using a random graph model, $G(n, p)$, where parameters n and p control the size of graph and the probability of two nodes connecting, respectively. We iterate from $n = 100$ to $n = 1000$ in steps of 100. p is set to $\log(n)/n$ for relatively sparse graphs and then moved to 0.1 and higher. Though we only report the sparse case

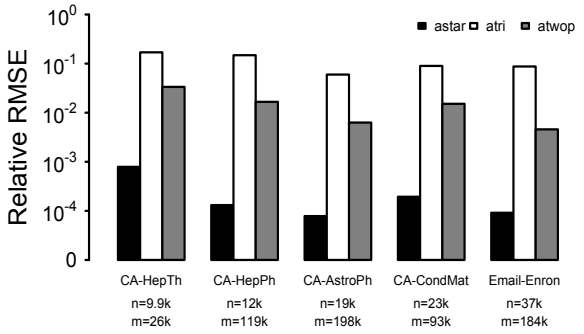


Figure 5.4: Perturbation error on real graphs

and $p = 0.1$, results for larger p agree with the conclusions. Error measurement is root mean square error (RMSE).

Alternating k-star

As described in Section 5.3.1, we can apply the Laplace mechanism (LAP) directly or compute the degree distribution privately first, by isotonic regression (ISO) from [45] and use it as a sufficient statistic for alternating k-star. Figure 5.1 shows the error of the two methods by varying p and λ, with different settings of $\epsilon = 1, 0.1$, listed in the legend text. As we do not have analytical RMSE for the ISO case, it is calculated from 100 independent perturbations. We clearly see LAP significantly outperforms ISO, even when $\lambda = 10$ at both ϵ settings (and recall that the global sensitivity is 2λ). For the rest of this section, if not stated, we set $\lambda = 2$ as it is the value normally recommended [71] and usually plays a minor part in the workflow.

Alternating k-triangle

The first-order local sensitivity bounding algorithm is applied here while setting ϵ to 1 and 0.1 and fixing $\delta = 0.01$. In Figure 5.2, we use "LSB" to represent Algorithm 1. For comparison purposes, we plot the *non-private* noisy output resulting from adding Laplace noise based on true local sensitivity, marked as "LS" in Figure 5.2. We find that LSB can add modest error when compared to this non-private baseline, especially when the privacy budget ϵ is relatively large.

Alternating k-twopath

We discussed in Section 5.3.3 how a first-order or second-order local sensitivity bound can be applied to alternating k-twopath. We present these results in Figure 5.3, by distinguishing them as "1-LSB" and "2-LSB". We find that for our test cases with random graphs, 1-LSB is consistantly better than 2-LSB, illustrated by RMSE in the left two subfigures. Referring to Sec. 5.3.3, recall that in 1-LSB we bound (5.8) while in 2-LSB we bound eqrefeq:ktwop1 by using bounded (5.9). If (5.7) and (5.9) are not small enough compared to (5.8), the fact that we split the budget of privacy one more time will outweigh the gain. In the right two subfigures of Figure 5.3, we plot the true local sensitivity and the expected values of private, bounded local sensitivity for both LSB algorithms. We see that 1-LSB results in a bound that is close to the true value but that 2-LSB results in a significant over-estimate, especially with a smaller $\epsilon = 0.1$. Although 1-LSB is superior across our tested networks, it remains possible that 2-LSB could outperform 1-LSB for particular input graphs or large ϵ and λ.

Real graphs

For real graphs, we consider several collected networks from the SNAP collection[2] to determine if our alternating statistics can be perturbed in a "meaningful" way, i.e., small relative noise that doesn't destroy utility. Our metric is *relative* RMSE, which is RMSE divided by the true statistic. As shown in Figure 5.4, with $\epsilon = 0.1$, all three alternating statistics (with shortened names: astar, atri, atwop) are estimated with low relative error. In particular, error for alternating k-star is between 10^{-3} and 10^{-4}, alternating k-triangle at 10^{-1} and alternating k-twopath at 10^{-2}.

[2]http://snap.stanford.edu

Network	nodes	edges	astar	atri	atwop
dolphins	62	159	418.1	177.5	705.4
lesmis	77	254	756.4	426.4	1565.5
polbooks	105	441	1355.4	715.5	2817.5
adjnoun	112	425	1292.9	452.1	3801.0
football	115	613	1992.4	922.3	3675.3

Table 5.1: Real networks for ERGM parameter estimation

Model	Model terms	Perturbation mech
M1	edges, astar	LAP, LAP
M2	edges, atri	LAP, 1-LSB
M3	edges, atwop	LAP, 1-LSB

Table 5.2: Model descriptions

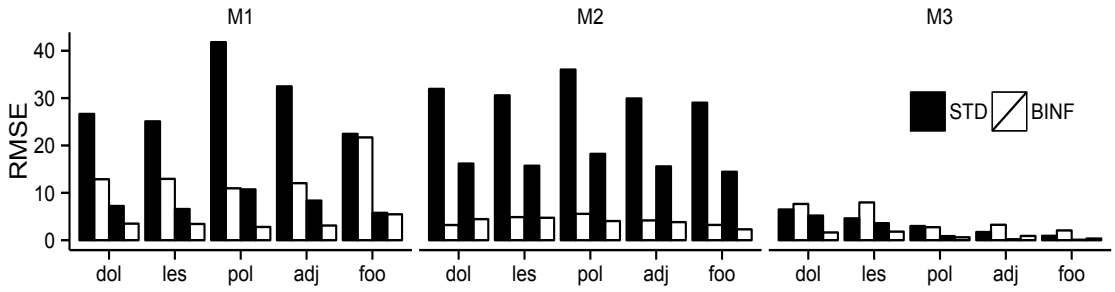


Figure 5.5: Parameter estimation with private statistics. Every four bars, from left to right, are $\theta_1, \theta_1, \theta_2, \theta_2$.

5.5.2 ERGM parameter estimation

For the evaluation of ERGM parameter estimation, we want to compare the algorithms in Section 5.4. In practice, the data owner will only perturb each statistic once and then release it to the analysts. As the perturbation is a randomized process, our goal is to understand how good our estimation algorithm is on *average*. So for each graph and each model description, we perturb the statistics $N = 50$ times and run the estimation algorithm on each perturbation, finally measuring their quality by RMSE with respect to estimates in the non-private case, $\sqrt{1/N \sum_{i \in [1,N]} (\hat{\theta}_i - \theta)^2}$, where θ is the "true" value, calculated from the non-private estimation algorithm from [50] or [14], $\hat{\theta}_i$ is θ from i-th perturbation.

As mentioned in [14], the estimation using the Bayesian technique has general scalability issues, where it becomes very slow for any graphs beyond a few hundred of nodes. Moreover such time cost also varies with the model terms, e.g., alternating k-twopath takes much more time than the other two alternating statistics, as calculation of the acceptance ratio in MCMC sampling of $x \sim p(x|\theta)$ is more complicated. Therefore, here we focus on smaller graphs, and this is the common practice for many ERGM works such as [14, 20, 71]. Our test networks[3] include dolphins, lesmis, polbooks, adjnoun and football. Detailed facts are listed in Table 5.1. We fix $\epsilon = 0.5$ and $\delta = 0.01$.

We experimented with three models, each of which corresponds to one of the alternating statistics, with the purpose of testing estimation by isolating other factors. We include the count of edges as a shared term in all models, as it is very common in ERGM applications. As shown in Table 5.2, each model contains two terms, with correspondingly two parameters, $\theta = (\theta_1, \theta_2)$. The estimation algorithms will be standard estimation (STD) and Bayesian inference (BINF). In all cases, the privacy

[3] http://www-personal.umich.edu/~mejn/netdata/

budget is distributed evenly in a way such that each generation of noise uses same share of the overall ϵ. In Figure 5.5, each graph is represented with 4 bars, showing θ_1 of STD, θ_1 of BINF, θ_2 of STD, θ_2 of BINF. In M1 and M2, we see a significant improvement of θ from STD to BINF. Especially in M2, BINF limits all errors to around 5 or smaller where STD can go up much higher. We believe this is because BINF can utilize the extra information presented by the local sensitivity bound as shown in Example 5.8. In M3, compared to the other models, we see that parameters of the model is quite insensitive to the changes due to perturbation, i.e., all graphs show much lower errors even under STD. In such situation, theoretically, there is not much room left for the improvement from BINF. This is illustrated in our experiment by showing comparable performance from both methods on M3. In general, we think BINF can improve the accuracy of parameter estimation significantly by leveraging the privacy distribution, while at the same time, the amount of benefit will vary depending on intrinsic properties of the model.

5.6 Related work

Differential privacy [25] has been actively studied in many sub-areas of computer science. Although the original focus was mainly on tabular data, the definition can be adapted to graph data [45] as well as other data models. Most research into differentially private analysis of graphs has focused on releasing graph statistics, e.g., degree sequence [45], triangle/star [55, 79], joint degree distribution/assortativity [83, 90] and clustering coefficient [109]. For modeling graphs privately, we are aware only of a private Kronecker graph modeling approach under differential privacy [75]. While our work relies on obtaining good private statistics, the ultimate goal is to allow ERGM modeling under differential privacy.

All of these works, including ours, protect relationships, i.e. they support edge-differential privacy. A stronger standard is to protect individuals, where neighbors

are defined by changing a single node. Recently, researchers have developed some mechanisms for calculating private graph statistics under node differential privacy [12, 16, 56].

Parameter estimation for ERGMs has also evolved from pseudo likelihood estimation (MPLE) [6], to Monte Carlo maximum likelihood (MC-MLE) [35] to recent stochastic approximation [99] and Bayesian inference [14]. These advances have helped ERGMs become central to social network analysis with many successful applications [71].

CHAPTER 6

CONCLUSION

This dissertation has addressed the challenges of sanitizing data for privacy purpose in the context of data sharing. Although we are looking at problems with various data sources, from tabular data to graph data, and various privacy requirements, from polices based privacy protection to differential privacy, we keep our discipline of designing a sanitization process consistent: protecting privacy while preserving as much utility as possible.

6.1 Review of contributions

In Chapter 3, we have presented a framework for limiting access to historical data, while still permitting auditing. Our redaction rules hide values but preserve information about the lifetime of tuples in a database, allowing an auditor to get accurate answers from the historical record despite the information removed by retention restrictions. We demonstrated that our techniques have a modest performance overhead, even when implemented using a standard relational system, and that the uncertainty introduced by sample retention policies is acceptable. By proposing two different models, we allow users to tune the system between accuracy and performance: TI gives you better performance but less accuracy while TC offers improved accuracy for audit queries under sanitized histories, at the expense of increased query processing complexity.

Chapter 4 has addressed untrusted analysts running accurate performance evaluation tasks without compromising privacy. Our method releases a differentially private

model of the database, allowing an analyst to sample synthetic databases consistent with the model. To achieve this we re-define differential privacy for multi-relations, and present novel techniques for selecting the model, perturbing statistics and sampling databases. To our knowledge, our framework is the only method for generating test databases while providing a rigorous guarantee of privacy for individuals in the database.

Chapter 5 has addressed estimating parameters for the exponential random graph model under differential privacy. Our solution decomposes the process into two steps: releasing private statistics first and running estimation second. The local sensitivity-based mechanism can offer lower error than common baselines. The redesigned Bayesian inference based parameter estimation is flexible and more accurate than standard methods. For future work, improving scalability is an interesting direction and the advance of technique in this area can both benefit our work and ERGM estimation in general.

6.2 Future directions

Auditing with more power We assume that retention policies are non-negotiable in Chapter 3, despite the auditors' interest in analysis tasks. This assumption could be reconsidered in the future work by prioritizing auditing accuracy, at the potential cost of retention policy secrecy. In addition, a compelling extension to our sanitization model could use generalization or summarization of values instead of redaction. This would impose some cost to confidentiality, but may significantly improve auditing capabilities. Currently, our preservation rules consist of tuple-level specifications. In the future we would like to integrate more complex view-based preservation rules, such as those considered by [4, 43], or rules targeting specific attribute values. We would like to investigate alternatives for supporting the periodic application of retention policies as a database evolves. And we would like to evaluate our system using data

histories based on well-known benchmark databases such as TPC-H, or using real data sets and workloads, as well as explore other physical organizations that could lead to improved performance.

Differential privacy in multi-table databases Our framework in Chapter 4 can be deployed using other differentially-private mechanisms, thus a natural future direction is to compare the utility achievable with different mechanisms. In addition, a side effect of protecting an entity in our multi-relation scenario is that the descendant entities are also protected. So it is interesting is to expand multi-relation differential privacy if general multiple entities need to be protected. We also hope our notion of multi-relation differential privacy, instead of bonded to key relationship, can be interpreted with general correlation among attributes.

Modeling graph under differential privacy Modeling graph under differential privacy is usually considered a more difficult problem than simply releasing private graph statistics. Our work in Chapter 5 is one of pioneered attempt in the area. In the future, we'd like to understand this problem in a bigger picture by considering more modeling techniques.

BIBLIOGRAPHY

[1] Antova, L., Jansen, T., Koch, C., and Olteanu, D. Fast and Simple Relational Processing of Uncertain Data. In *ICDE* (2008), pp. 983–992.

[2] Arasu, Arvind, Kaushik, Raghav, and Li, Jian. Data generation using declarative constraints. In *SIGMOD* (2011).

[3] ARMA Internaltional. Generally accepted recordkeeping principles. `www.arma.org/GARP/`.

[4] Ataullah, A.A., Aboulnaga, A., and Tompa, F.W. Records retention in relational database systems. In *CIKM* (2008), ACM, pp. 873–882.

[5] Bertino, Elisa, Bettini, Claudio, and Samarati, Pierangela. A temporal authorization model. In *ACM CCS Conference* (New York, NY, USA, 1994), ACM Press, pp. 126–135.

[6] Besag, Julian. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)* (1974).

[7] Binnig, C., Kossmann, D., and Lo, E. Reverse query processing. In *ICDE* (2007).

[8] Binnig, C., Kossmann, D., Lo, E., and Özsu, M.T. Qagen: generating query-aware test databases. In *SIGMOD* (2007).

[9] Biskup, Joachim. A foundation of codd's relational maybe-operations. *ACM Trans. Database Syst. 8* (1983), 608–636.

[10] Blakeley, J.A., Coburn, N., and Larson, P. Updating derived relations: detecting irrelevant and autonomously computable updates. *TODS 14*, 3 (1989), 369–4000.

[11] Blakeley, Jose A., Larson, Per-Ake, and Tompa, Frank Wm. Efficiently updating materialized views. *SIGMOD Rec. 15*, 2 (1986), 61–71.

[12] Blocki, Jeremiah, Blum, Avrim, Datta, Anupam, and Sheffet, Or. Differentially private data analysis of social networks via restricted sensitivity. In *ITCS* (2013).

[13] Bruno, N., and Chaudhuri, S. Flexible database generators. In *VLDB* (2005).

[14] Caimo, Alberto, and Friel, Nial. Bayesian inference for exponential random graph models. *Social Networks 33*, 1 (2011), 41–55.

[15] Chaudhuri, Surajit, and Narasayya, Vivek. Automating statistics management for query optimizers. *IEEE Transactions on Knowledge and Data Engineering 13* (January 2001), 7–20.

[16] Chen, Shixi, and Zhou, Shuigeng. Recursive mechanism: towards node differential privacy and unrestricted joins. In *SIGMOD* (2013).

[17] Chomicki, J. Temporal query languages: a survey. In *Temporal Logic: ICTL'94* (1994), vol. 827, pp. 506–534.

[18] Cormode, G., Procopiuc, M., Shen, E., Srivastava, D., and Yu, T. Differentially private spatial decompositions. *ICDE* (2012).

[19] Cormode, Graham, Procopiuc, Cecilia, Srivastava, Divesh, and Tran, Thanh TL. Differentially private summaries for sparse data. In *ICDT* (2012).

[20] Cranmer, Skyler J, and Desmarais, Bruce A. Inferential network analysis with exponential random graph models. *Political Analysis 19*, 1 (2011), 66–86.

[21] Ding, Bolin, Winslett, Marianne, Han, Jiawei, and Li, Zhenhui. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD* (2011).

[22] Doucet, Arnaud, Godsill, Simon J, and Robert, Christian P. Marginal maximum a posteriori estimation using markov chain monte carlo. *Statistics and Computing* (2002).

[23] Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. Our data, ourselves: Privacy via distributed noise generation. *EUROCRYPT* (2006).

[24] Dwork, C., and Lei, J. Differential privacy and robust statistics. In *Proceedings of the 41st annual ACM symposium on Theory of computing* (2009), ACM, pp. 371–380.

[25] Dwork, C., McSherry, F., Nissim, K., and Smith, A. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography* (2006).

[26] EMC Corporation. `www.emc.com`.

[27] Ercegovac, V., DeWitt, D.J., and Ramakrishnan, R. The texture benchmark: measuring performance of text queries on a relational dbms. In *VLDB* (2005).

[28] Fabbri, D., LeFevre, K., and Zhu, Q. PolicyReplay: misconfiguration-response queries for data breach reporting. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 36–47.

[29] Fienberg, S.E., Slavkovic, A., and Uhler, C. Privacy preserving gwas data sharing. In *ICDMW* (2011).

[30] Gadia, Shashi K. A homogeneous relational model and query languages for temporal databases. *ACM Trans. Database Syst. 13* (1988), 418–448.

[31] Gadia, Shashi K., Nair, Sunil S., and Poon, Yiu-Cheong. Incomplete information in relational temporal databases. In *18th VLDB Conference* (1992).

[32] Gamerman, Dani, and Lopes, Hedibert F. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*, vol. 68. CRC Press, 2006.

[33] Garcia-Molina, Hector, Labio, Wilburt, and Yang, Jun. Expiring data in a warehouse. In *VLDB Conference* (1998), pp. 500–511.

[34] Getoor, Lise, Taskar, Benjamin, and Koller, Daphne. Selectivity estimation using probabilistic models. In *SIGMOD* (2001).

[35] Geyer, Charles J, and Thompson, Elizabeth A. Constrained monte carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society.* (1992).

[36] Goodreau, Steven M, Kitts, James A, and Morris, Martina. Birds of a feather, or friend of a friend? using exponential random graph models to investigate adolescent social networks*. *Demography 46*, 1 (2009), 103–125.

[37] Grahne, G. *The Problem of Incomplete Information in Relational Databases.* Springer, 1991.

[38] Gray, J., Sundaresan, P., Englert, S., Baclawski, K., and Weinberger, P.J. Quickly generating billion-record synthetic databases. In *SIGMOD* (1994).

[39] GRM LLC. `www.grmdocumentmanagement.com`.

[40] Guo, S., Sun, W., and Weiss, M.A. Solving satisfiability and implication problems in database systems. *ACM Transactions on Database Systems (TODS) 21*, 2 (1996), 270–293.

[41] Gupta, V., Miklau, G., and Polyzotis, N. Private database synthesis for outsourced system evaluation. *5th Alberto Mendelzon International Workshop on Foundations of Data Management* (2011).

[42] Hardt, M., Ligett, K., and McSherry, F. A simple and practical algorithm for differentially private data release. In *NIPS* (2012).

[43] Hasan, R., and Winslett, M. Trustworthy vacuuming and litigation holds in long-term high-integrity records retention. In *Proceedings of the 13th International Conference on Extending Database Technology* (2010), ACM, pp. 621–632.

[44] Hasan, R., Winslett, M., and Mitra, S. Efficient Audit-based Compliance for Relational Data Retention. *UIUC Dept. of CS Tech Report UIUCDCS-R-2009-3044* (2009).

[45] Hay, M., Li, C., Miklau, G., and Jensen, D. Accurate estimation of the degree distribution of private networks. In *ICDM* (2009).

[46] Hay, Michael, Rastogi, Vibhor, Miklau, Gerome, and Suciu, Dan. Boosting the accuracy of differentially-private histograms through consistency. *VLDB* (2010).

[47] Hochbaum, D.S., and Moreno-Centeno, E. The inequality-satisfiability problem. *Operations Research Letters 36*, 2 (2008), 229–233.

[48] Houkjær, K., Torp, K., and Wind, R. Simple and realistic data generation. In *VLDB* (2006).

[49] Hunter, D.R. Curved exponential family models for social networks. *Social Networks 29*, 2 (2007), 216–230.

[50] Hunter, D.R., and Handcock, M.S. Inference in curved exponential family models for networks. *Journal of Computational and Graphical Statistics* (2006).

[51] Imielinski, Tomasz, and Lipski, Witold. Incomplete information in relational databases. *J. ACM 31*, 4 (1984), 761–791.

[52] Jensen, C. S., and Mark, L. Queries on change in an extended relational model. *IEEE TKDE 4* (1992), 192–200.

[53] Jensen, C. S., Mark, L., and Roussopoulos, N. Incremental implementation model for relational databases with transaction time. *IEEE Trans. Knowl. Data Eng. 3* (1991), 461–473.

[54] Johansen, Adam M, Doucet, Arnaud, and Davy, Manuel. Particle methods for maximum likelihood estimation in latent variable models. *Statistics and Computing* (2008).

[55] Karwa, V., Raskhodnikova, S., Smith, A., and Yaroslavtsev, G. Private analysis of graph structure. In *VLDB* (2011).

[56] Kasiviswanathan, Shiva Prasad, Nissim, Kobbi, Raskhodnikova, Sofya, and Smith, Adam. Analyzing graphs with node differential privacy. In *TCC* (2013).

[57] Kifer, D., and Machanavajjhala, A. No free lunch in data privacy. In *SIGMOD* (2011).

[58] Koubarakis, M. Database models for infinite and indefinite temporal information. *Information Systems 19* (1994), 141.

[59] Lageweg, BJ, Lenstra, JK, and Kan, A.H.G.R. Minimizing maximum lateness on one machine: computational experience and some applications. *Statistica Neerlandica 30*, 1 (1976), 25–41.

[60] LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., and DeWitt, D. Limiting disclosure in hippocratic databases. *VLDB Conference* (2004), 108–119.

[61] Leskovec, Jure, Chakrabarti, Deepayan, Kleinberg, Jon, Faloutsos, Christos, and Ghahramani, Zoubin. Kronecker graphs: An approach to modeling networks. *JMLR* (2010).

[62] LexisNexis. Document retention & destruction policies for digital data.

[63] Li, C., Hay, M., Rastogi, V., Miklau, G., and McGregor, A. Optimizing linear counting queries under differential privacy. In *PODS* (2010).

[64] Li, Chao, and Miklau, Gerome. An adaptive mechanism for accurate query answering under differential privacy. In *VLDB* (2012).

[65] Lo, E., Cheng, N., and Hon, W.K. Generating databases for query workloads. *VLDB* (2010).

[66] Lomet, David B., Barga, Roger S., Mokbel, Mohamed F., Shegalov, German, 0002, Rui Wang, and Zhu, Yunyue. Transaction time support inside a database engine. In *ICDE* (2006), p. 35.

[67] Lu, W., and Miklau, G. Auditguard: a system for database auditing under retention restrictions. *VLDB* (2008).

[68] Lu, W., and Miklau, G. Auditing a database under retention restrictions. In *ICDE* (2009).

[69] Lu, W., Miklau, G., and Immerman, N. Auditing a database under retention policies. *The VLDB Journal* (2013).

[70] Lu, Wentian, Miklau, Gerome, and Gupta, Vani. Synthesizing databases privately for untrusted system evaluation. In *ICDE* (2014).

[71] Lusher, D., Koskinen, J., and Robins, G. *Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 2012.

[72] McSherry, F., and Mahajan, R. Differentially-private network trace analysis. In *SIGCOMM* (2010).

[73] McSherry, F., and Mironov, I. Differentially private recommender systems: building privacy into the net. In *SIGKDD* (2009).

[74] McSherry, Frank D. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD* (2009).

[75] Mir, Darakhshan, and Wright, Rebecca N. A differentially private estimator for the stochastic kronecker graph model. In *EDBT/ICDT Workshops* (2012).

[76] Mullins, Craig S. Database archiving for long-term data retention. `http://www.tdan.com/view-articles/4591`, 2006.

[77] Murray, Iain, Ghahramani, Zoubin, and MacKay, David. Mcmc for doubly-intractable distributions. *arXiv* (2012).

[78] Newman, Mark. *Networks: an introduction*. Oxford University Press, 2010.

[79] Nissim, Kobbi, Raskhodnikova, Sofya, and Smith, Adam. Smooth sensitivity and sampling in private data analysis. In *STOC* (2007).

[80] OpenText Corporation. `www.opentext.com`.

[81] Oracle. Oracle database security guide (11.1).

[82] Perez, Rocio Aldeco, and Moreau, Luc. Securing provenance-based audits. In *International Provenance and Annotation Workshop 2010* (July 2010), Springer.

[83] Proserpio, Davide, Goldberg, Sharon, and McSherry, Frank. Calibrating data to sensitivity in private data analysis. In *VLDB* (2014).

[84] RainStor Inc. `rainstor.com`.

[85] Ramakrishnan, Raghu, and Gehrke, Johannes. *Database management systems*. Osborne/McGraw-Hill, 2000.

[86] Rastogi, Vibhor, Hay, Michael, Miklau, Gerome, and Suciu, Dan. Relationship privacy: output perturbation for queries with joins. In *PODS* (2009).

[87] Rebecca Bond, Kevin Yeung-Kuen See, Carmen Ka Man Wong Yuk-Kuen Henry Chan Yuk-Kuen Henry Chan. *Understanding DB2 9 Security*. IBM Press, 2006.

[88] Robins, G., Snijders, T., Wang, P., Handcock, M., and Pattison, P. Recent developments in exponential random graph p* models for social networks. *Social networks* (2007).

[89] Rosenkrantz, D. J., and Hunt, H. B. Processing conjunctive predicates and queries. In *VLDB Conference* (1980), p. 72.

[90] Sala, A., Zhao, X., Wilson, C., Zheng, H., and Zhao, B.Y. Sharing graphs using differentially private graph models. In *SIGCOMM* (2011).

[91] SAND Technology. `www.sand.com`.

[92] Sarda, N. L. Extensions to sql for historical databases. *IEEE Trans. Knowl. Data Eng. 2* (1990), 220–230.

[93] Sarma, A.D., Benjelloun, O., Halevy, A., and Widom, J. Working models for uncertain data. In *ICDE* (2006).

[94] Schneier, B., and Kelsey, J. Secure Audit Logs to Support Computer Forensics. *ACM Trans. Inf. Syst. Secur. 2*, 2 (1999), 159–176.

[95] Seltzer, M., Krinsky, D., Smith, K., and Zhang, X. The case for application-specific benchmarking. In *Hot Topics in OS* (1999).

[96] Shaull, Ross, Shrira, Liuba, and Xu, Hao. Skippy: a new snapshot indexing method for time travel in the storage manager. In *ACM SIGMOD Conference* (2008), pp. 637–648.

[97] Simons, B., and Sipser, M. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research* (1984), 80–88.

[98] Skyt, J., Jensen, C.S., and Mark, L. A foundation for vacuuming temporal databases. *Data & Knowl. Eng. 44*, 1 (2003), 1–29.

[99] Snijders, T.A.B. Markov chain monte carlo estimation of exponential random graph models. *Journal of Social Structure* (2002).

[100] Snijders, Tom AB, Pattison, Philippa E, Robins, Garry L, and Handcock, Mark S. New specifications for exponential random graph models. *Sociological methodology* (2006).

[101] Snodgrass, R. T. *Developing time-oriented database applications in SQL*. Morgan Kaufmann Publishers Inc., USA, 1999. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

[102] Snodgrass, Richard T., and Collberg, Christian S. *The τ-BerkeleyDB Temporal Subsystem*. Published: Available at `www.cs.arizona.edu/tau/tbdb/`.

[103] Snodgrass, Richard Thomas. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[104] Snodgrass, R.T., Yao, S.S., and Collberg, C. Tamper detection in audit logs. In *13th VLDB Conference* (2004), pp. 504–515.

[105] Stahlberg, Patrick, Miklau, Gerome, and Levine, Brian Neil. Threats to privacy in the forensic analysis of database systems. In *SIGMOD Conference* (2007), pp. 91–102.

[106] Tay, Y., Dai, B., Wang, T., Sun, Y., Lin, Y., and Lin, Y. Upsizer: Synthetically scaling an empirical relational database. Tech. rep., Nat. Univ. of Singapore, 2010.

[107] Toman, D. Expiration of historical databases. In *Symposium on Temporal Representation and Reasoning (TIME)* (2001), pp. 128–135.

[108] Wang, Qihua, Yu, Ting, Li, Ninghui, Lobo, Jorge, Bertino, Elisa, Irwin, Keith, and Byun, Ji-Won. On the correctness criteria of fine-grained access control in relational databases. In *VLDB Conference* (2007), pp. 555–566.

[109] Wang, Yue, Wu, Xintao, Zhu, Jun, and Xiang, Yang. On learning cluster coefficient of private networks. *Social network analysis and mining* (2013).

[110] Waters, B.R., Balfanz, D., Durfee, G., and Smetters, D.K. Building an encrypted and searchable audit log. In *NDSS* (2004), vol. 6.

[111] Weiss, Neil A, Holmes, Paul T, and Hardy, Michael. *A course in probability*. Pearson Addison Wesley, 2006.

[112] Wrozek, Brian. Electronic Data Retention Policy, 2001.

[113] Wu, X., Wang, Y., Guo, S., and Zheng, Y. Privacy preserving database generation for database application testing. *Fundamenta Informaticae 78*, 4 (2007), 595–612.

[114] Xiao, X., Bender, G., Hay, M., and Gehrke, J. ireduct: Differential privacy with reduced relative errors. In *SIGMOD* (2011).

[115] Xiao, X., Wang, G., and Gehrke, J. Differential privacy via wavelet transforms. *ICDE* (2010).

[116] Yaroslavtsev, Grigory, Cormode, G, Procopiuc, CM, and Srivastava, D. Accurate and efficient private release of datacubes and contingency tables. ICDE.

[117] Yuan, Ganzhao, Zhang, Zhenjie, Winslett, Marianne, Xiao, Xiaokui, Yang, Yin, and Hao, Zhifeng. Low-rank mechanism: Optimizing batch queries under differential privacy. VLDB, 2012.

[118] ZL Technologies, Inc. `www.zlti.com`.

[119] ZyLAB. `www.zylab.com`.