Generating Private Synthetic Databases for Untrusted System Evaluation

Wentian Lu, Gerome Miklau, Vani Gupta
School of Computer Science, University of Massachusetts Amherst
{wen,miklau,vani}@cs.umass.edu

***Abstract*—Evaluating the performance of database systems is crucial when database vendors or researchers are developing new technologies. But such evaluation tasks rely heavily on actual data and query workloads that are often unavailable to researchers due to privacy restrictions. To overcome this barrier, we propose a framework for the release of a synthetic database which accurately models selected performance properties of the original database. We improve on prior work on synthetic database generation by providing a formal, rigorous guarantee of privacy. Accuracy is achieved by generating synthetic data using a carefully selected set of statistical properties of the original data which balance privacy loss with relevance to the given query workload. An important contribution of our framework is an extension of standard differential privacy to multiple tables.**

I. 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 [17], [4], [27], [1], [3], [21] 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], [11], [1]. 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 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'.

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.

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

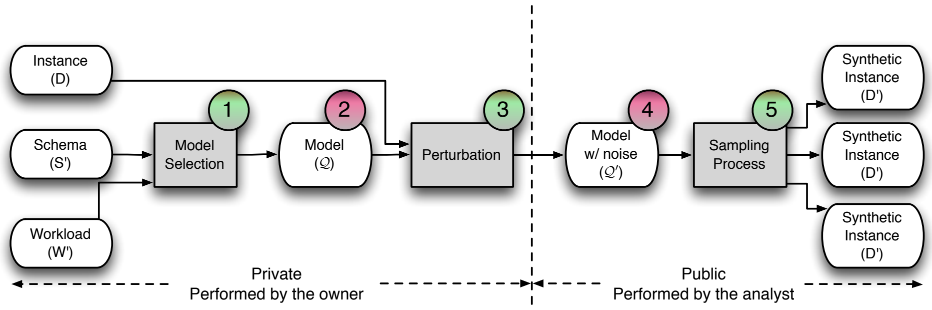


Fig. 1. Our Approach: the *owner* selects (procedure of box 1, Section III) a model $\mathcal{Q}$ (rounded box 2) given schema and workload. A model contains a set of carefully chosen queries, and their answers (statistics) can be calculated with instance (D). The owner now perturbs (procedure of box 3, Section V) 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 VI) one or more synthetic instances.

database generation approaches [2], [1], [4], [21], [17] do not offer any formal privacy guarantees. Instead, they 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 [1] acknowledge the privacy issues of releasing cardinality information during data generation. One exception is the work of Wu et al. [28], 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.

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.

II. Preliminaries

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

A. 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 paper, we focus on workload queries involving joins only on key attributes. This assumption is also accepted by the literature (e.g. [1]) and it actually covers a wide range of applications, including TPC-H benchmark.

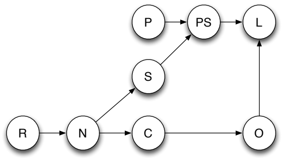


Fig. 2. The schema of TPC-H represented as a directed graph.

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 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.

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:

$$\begin{aligned} q_1 &: \quad |\sigma_{\mathsf{C}.gender=M}(\mathsf{C})| \\ q_2 &: \quad |\sigma_{\mathsf{C}.gender=M}(\mathsf{C} \bowtie \mathsf{O})| \end{aligned}$$

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. 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.

B. The differential privacy guarantee

An algorithm is differentially private if its output is statistically close on two database inputs that differ by one record. Two such databases are called *neighbors*.

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

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

If $\delta = 0$, $\mathcal{K}$ is ϵ-differentially private, according to the standard definition. Otherwise, $\mathcal{K}$ is (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. 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], [11], [1].

In Section IV 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 [12], location data [7] and network traces [22].

C. *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*. The sensitivity of a query is the maximum possible difference in the output when evaluating the query on two neighboring databases.

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 reason 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 [30], [16], [19], [31], [8], [32], [15], [6]. 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 [20], based on the matrix mechanism [19], 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.

III. 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.

A. *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 [1]. 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

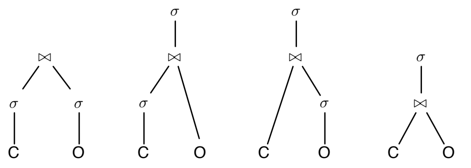


Fig. 3. Possible query trees for $\sigma_{\mathsf{C}.gender=M \wedge \mathsf{O}.amount>100}(\mathsf{C} \bowtie \mathsf{O})$

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

$$\begin{aligned} w_1 &: \sigma_{\mathsf{C}.gender=M \wedge \mathsf{O}.year=2010}(\mathsf{C} \bowtie \mathsf{O}) \\ w_2 &: \sigma_{\mathsf{C}.age=40}(\mathsf{C}) \end{aligned}$$

$\Gamma(w_1)$ 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 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.

B. 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:

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

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.

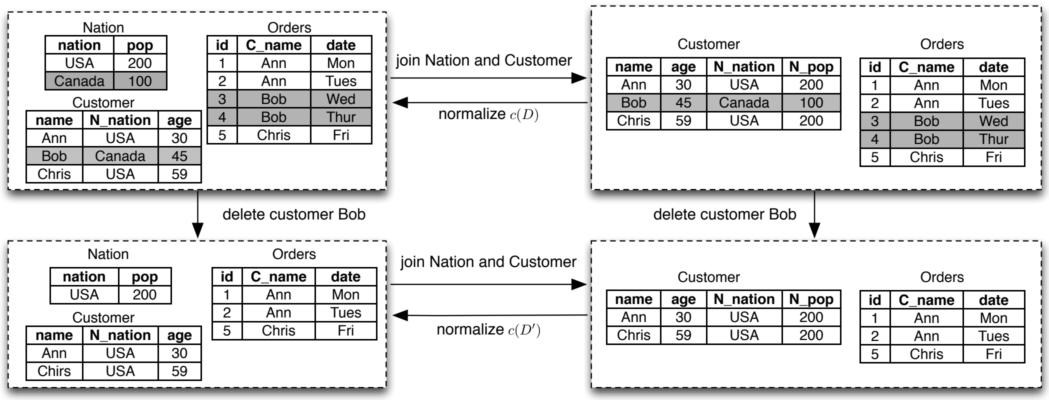
IV. Differential privacy for multiple-relation databases

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 [18]. 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 [24] 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 relationships. We then discuss the calculation of query sensitivity and the calculation of sensitivity for the queries that make up a model.

A. 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



Nation

nation	pop
USA	200
Canada	100

Customer

name	N_nation	age
Ann	USA	30
Bob	Canada	45
Chris	USA	59

Orders

id	C_name	date
1	Ann	Mon
2	Ann	Tues
3	Bob	Wed
4	Bob	Thur
5	Chris	Fri

Customer

name	age	N_nation	N_pop
Ann	30	USA	200
Bob	45	Canada	100
Chris	59	USA	200

Orders

id	C_name	date
1	Ann	Mon
2	Ann	Tues
3	Bob	Wed
4	Bob	Thur
5	Chris	Fri

Nation

nation	pop
USA	200

Customer

name	N_nation	age
Ann	USA	30
Chirs	USA	59

Orders

id	C_name	date
1	Ann	Mon
2	Ann	Tues
5	Chris	Fri

Customer

name	age	N_nation	N_pop
Ann	30	USA	200
Chris	59	USA	200

Orders

id	C_name	date
1	Ann	Mon
2	Ann	Tues
5	Chris	Fri

Fig. 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.

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 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 4.1 (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 4.1 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 2: 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 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. Due to the lack of space, we omit the details of that.

B. *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):

$$\Delta_q \cdot df(v(q)) \tag{1}$$

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

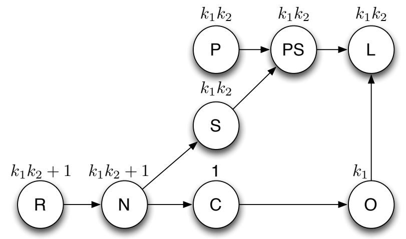


Fig. 5. Difference (*df* value) between neighboring TPC-H instances.

in table Y for the key/foreign-key relationship $X \rightarrow Y$. 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 IV.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 IV.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:         $df(X) = \sum_{Y \rightarrow X} \mathcal{K}(Y, X) df(Y)$
5:     else   $df(X) = 0$
6: for $X$ in reverse topological order of $G_S$ do
7:     $df(X) = df(X) + \sum_{X \rightarrow Y}[df(Y) - \mathcal{K}(X, Y) df(X)]$
   return all $df$ values
```

Example 3: 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. 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:

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

For example, in Figure 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 IV-A 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.

V. 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.

A. 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 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 6(c)) is the query matrix containing the three counting queries of Figure 6(a) based on $\mathbf{x}$ in Figure 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 [9], can be defined as:

Definition 5.1 (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 IV-B), 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.

B. 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 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: Consider the saturated model for workload queries W in Example 1 (Section III-A). 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 $\dim(\mathbf{x}_\mathsf{O}) =$

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 $\mathbf{Q}\mathbf{x}$.

Fig. 6. An example of counting queries and a data vector.

$\{\mathsf{C}.gender, \mathsf{O}.year\}$. 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 $|\sigma_{\mathsf{C}.gender=M, \mathsf{C}.age=*}(\mathsf{C})|$.

C. 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 [19]. 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.)

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 5.2 (Matrix Mechanism): [19] 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:

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

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 [20] 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 $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n\}$ matched with data vectors $\mathcal{X}$ and $\mathcal{Q}$.

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 [20], 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 [9], [24], [23], also apply to the matrix mechanism, for the first time.

Proposition 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 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 2 inside each group and Proposition 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 1 only) and the fully joined one I_j with one group that contains all query matrices (applying Proposition 2 only).

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

$$\textsc{MinErr}_{\mathcal{M}}(I) = \min \sum_{\text{group } g\in I} \textsc{ErrG}_{\mathcal{M}}(g, \epsilon_g, \delta_g) \tag{3}$$

From Equation (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 $b = \mathrm{tr}(\mathbf{Q}(\mathbf{A}^t\mathbf{A})^{-1}\mathbf{Q}^t)$. Let $\Delta_{g(\mathcal{A})}$ be the sensitivity of group g's strategy matrices.

$$\begin{aligned}\textsc{ErrG}_{\mathcal{M}}(g, \epsilon_g, \delta_g) &= \sum_{i\in g} \frac{2\ln(2/\delta_g)}{\epsilon_g^2} \Delta^2_{g(\mathcal{A})}\, b_i \\ &= \frac{2\ln(2/\delta_g)}{\epsilon_g^2} (\sum_{i\in g} \Delta^2_{\mathbf{A}_i})(\sum_{i\in g} b_i) \end{aligned} \tag{4}$$

To calculate $\textsc{MinErr}_{\mathcal{M}}(I)$, we apply Lagrange multiplier to solve the optimization problem with objective function Equation (3) and two equality constraints $\sum_g \epsilon_g = \epsilon$ and $\sum_g \delta_g = \delta$, which gives us the following results:

$$\textsc{MinErr}_{\mathcal{M}}(I) = \frac{2}{\epsilon^2} \left(\sum_{\text{group } g\in I} \sqrt[3]{\ln(\frac{2}{\delta_g}) \cdot b_g c_g} \right)^3 \tag{5}$$

Here, the group trace value, b_g, is defined as $\sum_{\mathbf{Q}_i, \mathbf{A}_i \in g} \mathrm{tr}(\mathbf{Q}_i \ \ ({\mathbf{A}_i}^t \mathbf{A}_i)^{-1} \ \ \mathbf{Q}_i^t)$. 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 5: 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) 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 (5). 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

$\mathbf{x}_1$	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) $\mathbf{x}_{\mathsf{O}}$

Fig. 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.

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.

VI. 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 7. The idea of consistent data vectors discussed in Section V-B 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. [1]). 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 7, $\mathbf{x}_{\mathsf{O}}$[O.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 7, $C_{\mathsf{C}}^{\mathsf{O}} = \{\mathsf{C.age}\}$. If $\dim(\mathbf{x}_{\mathsf{O}})$ also includes N.nation, $E_{\mathsf{C}}^{\mathsf{O}} = \{\mathsf{N.nation}\}$ because the dimension N.nation is not in $\mathbf{x}_{\mathsf{C}}$ and is an attribute of N, an ancestor of C.

Theorem 1: 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 7, the violation happens because

$$\mathbf{x}_{\mathsf{C}}[\mathsf{C.age} \leq 40] \ngeq \lceil \frac{1}{\mathcal{K}(\mathsf{C},\mathsf{O})} \cdot \mathbf{x}_{\mathsf{O}}[\mathsf{C.age} \leq 40] \rceil$$

In the theorem above, we use propagation constraints defined in Section IV-A 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 VII-C). The idea is that whenever the inequality in Theorem 1 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.

VII. 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.

A. 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 [5], 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.

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

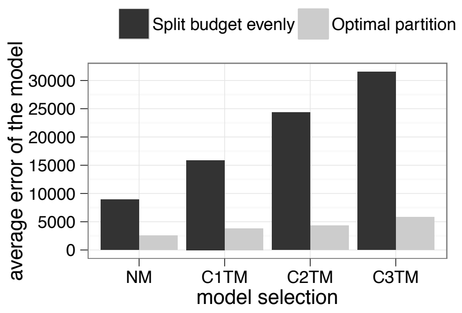


Fig. 8. Comparison of the error rates for different choices of privacy budget distribution

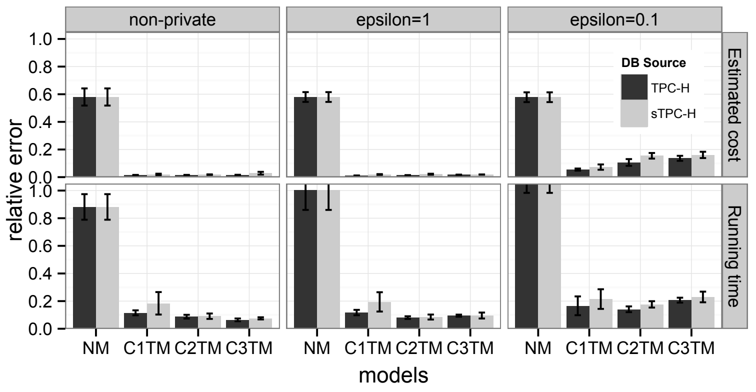


Fig. 9. Model Error and perturbation error of TPC-H and sTPC-H

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 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 (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 $\delta = 0.001$.

B. Modeling

We implement the model family described in Section III-B. The null model (NM) serves as a baseline approach because it does not depend on the workload. Table I 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 8 tells that our budget allocation algorithm can reach much lower total error than simply splitting budget evenly.

C. Sampling

The sampling process involves two steps: realization and sampling. We apply the approximated realization introduced in Section VI, avoiding expensive quadric programming. In the last row of Table I 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 I
DETAILED INFORMATION ABOUT MODELS

D. 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 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 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

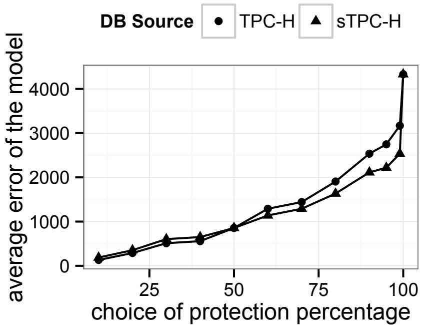


Fig. 10. Influence of protection percentage on error rates of model C2TM

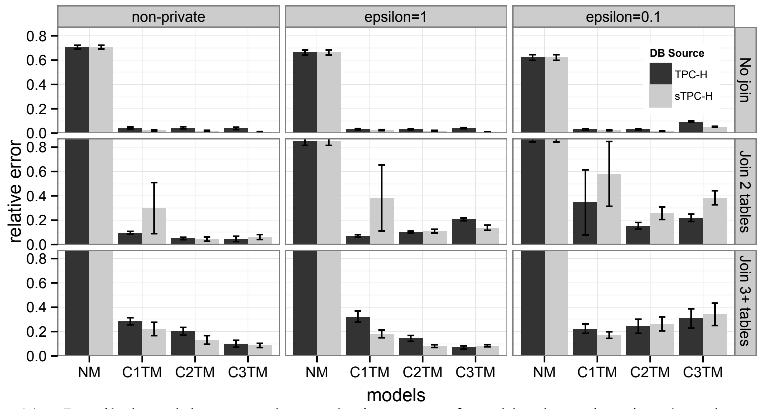


Fig. 11. Detailed model error and perturbation error of workload running time based on query complexity

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 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 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 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 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.

VIII. Related Work

There have been many proposed methods for synthesizing relational databases [13], [17], [4], [27], [1], [2], [3], [21]. Privacy is a commonly-cited motivation [27], [1]. Yet only one paper actually specifies a privacy condition for generated data [28] 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], [17], [4], [27] (i.e., intended to support any set of queries considered) or *workload-aware* [1], [3], [21] (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. [26] 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 [4], [3] have used cardinality statistics for (non-privately) synthesizing a database.

Differential privacy [10] 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 [16], [32], [29], [30]. 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. [19], [20] 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 paper are a significant extension to preliminary work [14]. The PINQ framework [24] 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 [25] consider queries with joins and show that certain limiting assumptions about the adversary can result in improved utility under a model of adversarial privacy.

IX. Conclusion

We consider the problem of allowing untrusted analysts to run 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.

Because our framework can be deployed using other differentially-private mechanisms, a natural future direction is to compare the utility achievable with different mechanisms. A side effect of protecting the C entity in our multi-relation scenario is that the descendant entities (O and L) are also protected. Another future direction is to expand multi-relation differential privacy if multiple entities other than C and its descendants need to be protected.

Acknowledgements: The authors are grateful to the anonymous reviewers for their feedback. The authors were supported by NSF IIS-0964094 and IIS-0643681. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *SIGMOD*, 2011.
[2] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In *ICDE*, 2007.
[3] C. Binnig, D. Kossmann, E. Lo, and M. Özsu. Qagen: generating query-aware test databases. In *SIGMOD*, 2007.
[4] N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, 2005.
[5] S. Chaudhuri and V. Narasayya. Automating statistics management for query optimizers. *IEEE Transactions on Knowledge and Data Engineering*, 13:7–20, January 2001.
[6] G. Cormode, C. Procopiuc, D. Srivastava, and T. T. Tran. Differentially private summaries for sparse data. In *ICDT*, 2012.
[7] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. *ICDE*, 2012.
[8] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, 2011.
[9] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. *EUROCRYPT*, 2006.
[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography*, 2006.
[11] V. Ercegovac, D. DeWitt, and R. Ramakrishnan. The texture benchmark: measuring performance of text queries on a relational dbms. In *VLDB*, 2005.
[12] S. Fienberg, A. Slavkovic, and C. Uhler. Privacy preserving gwas data sharing. In *ICDMW*, 2011.
[13] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. Weinberger. Quickly generating billion-record synthetic databases. In *SIGMOD*, 1994.
[14] V. Gupta, G. Miklau, and N. Polyzotis. Private database synthesis for outsourced system evaluation. *5th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2011.
[15] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, 2012.
[16] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially-private histograms through consistency. *VLDB*, 2010.
[17] K. Houkjær, K. Torp, and R. Wind. Simple and realistic data generation. In *VLDB*, 2006.
[18] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD*, 2011.
[19] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, 2010.
[20] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. In *VLDB*, 2012.
[21] E. Lo, N. Cheng, and W. Hon. Generating databases for query workloads. *VLDB*, 2010.
[22] F. McSherry and R. Mahajan. Differentially-private network trace analysis. In *SIGCOMM*, 2010.
[23] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the net. In *SIGKDD*, 2009.
[24] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.
[25] V. Rastogi, M. Hay, G. Miklau, and D. Suciu. Relationship privacy: output perturbation for queries with joins. In *PODS*, 2009.
[26] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The case for application-specific benchmarking. In *Hot Topics in OS*, 1999.
[27] Y. Tay, B. Dai, T. Wang, Y. Sun, Y. Lin, and Y. Lin. Upsizer: Synthetically scaling an empirical relational database. Technical report, Nat. Univ. of Singapore, 2010.
[28] X. Wu, Y. Wang, S. Guo, and Y. Zheng. Privacy preserving database generation for database application testing. *Fundamenta Informaticae*, 78(4):595–612, 2007.
[29] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: Differential privacy with reduced relative errors. In *SIGMOD*, 2011.
[30] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *ICDE*, 2010.
[31] G. Yaroslavtsev, G. Cormode, C. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. ICDE, 2013.
[32] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: Optimizing batch queries under differential privacy. VLDB, 2012.