Private Database Synthesis for Outsourced System Evaluation

Vani Gupta[1], Gerome Miklau[1], and Neoklis Polyzotis[2]

[1] Dept. of Computer Science, University of Massachusetts, Amherst, MA, USA
[2] Dept. of Computer Science, University of California, Santa Cruz, CA, USA

Abstract. The goal of this paper is to permit secure outsourced system evaluation. We propose a method for generating synthetic databases and obfuscating a workload of queries in order to protect the sensitive information present in the database. The synthetic database and workload can be used by a third party to accurately carry out performance tuning, index selection, or other system evaluation tasks. As a result, an untrusted third party can evaluate whether a new technology would benefit the data owner without the risk of a privacy breach.
Our approach is to employ state-of-the-art privacy mechanisms to compute the sufficient statistics of a statistical model of the true database. These statistics are safe to release, so a third party can then use them to generate one or more synthetic databases to be used as a surrogate for the true database.

1 Introduction

Enterprises managing large databases commonly need to perform system evaluation tasks. These include tuning the performance of existing systems (e.g. through physical design, logical design, index selection, or optimizer tweaking) or exploring potential benefits of adopting new systems or architectures (e.g. moving from a row-store to a column-store, or moving from a centralized to a parallel system). These system evaluation tasks are typically performed only by trusted parties employed by the enterprise because enterprise data and associated query workloads are too sensitive to reveal.

The goal of this paper is to permit secure outsourced system evaluation – that is, to allow these system evaluation tasks to be carried out accurately and safely by an untrusted party. This may be economically advantageous, for example, when an enterprise seeks to outsource performance tuning, or to simplify commercial software evaluation. Or it may benefit scientists, for instance, when a researcher wants to evaluate new database technology on realistic data.

It is currently very difficult to achieve the benefits of outsourced system evaluation. One possible solution is for the researcher or vendor to provide the owner with a working prototype system so the owner can deploy it using the real enterprise data. But this puts the burden of system setup on the owner and limits the ability of the vendor or researcher to customize the system. The researcher or vendor could also test on a benchmark database and workload (such as a TPC variant), designed to reflect properties commonly found in real applications. This permits the researcher to perform system setup and report results to the data owner, but the results may not be convincing since they do not reflect the particular properties of the owner's workload. In fact, we argue that benchmarks are frequently misused in experimental evaluation because realistic database workloads are unavailable. Benchmarks allow for the fair comparison of performance properties across systems. They are intended to be representative of some reasonably realistic scenario, but they do not represent the peculiar characteristics of real applications.

The goal of our work is therefore to construct, based on the real data of an enterprise, a synthetic database instance and query workload that: (1) can be safely released because it does not reveal sensitive information about enterprise data or proprietary practices; and (2) accurately reflects the physical and statistical properties of the database and workload so that system evaluation can be carried out on the synthetic data in place of the real data. In addition to supporting safe outsourced system evaluation, such techniques could be used to create a public repository of database instances and workloads to benefit the research community.

We focus on supporting system evaluation tasks which are determined by a relational database instance (tables and their contents) and a workload (a set of relational queries). For example, the task of automatic index selection algorithm involves computing the set of indexes which will result in the lowest aggregate query execution time for a given database and workload. Similarly, a new component of a query optimizer is evaluated by comparing performance on a given database and workload.

To achieve our goal we need to ensure that sensitive information present in the database and workload is protected. As schemas are likely to be the least sensitive, we assume that the release of a schema isomorphic to the original, with obfuscated attribute names, is acceptable to the enterprise. We rename tables and columns, remove detailed domain information, but preserve key-foreign key relationships and general data types. We transform the workload of queries to the new schema, and obfuscate constants appearing in queries. Lastly, the actual contents of the database are the most sensitive as they may contain credit card numbers, personal information, financial records, etc. We use the formal standard of differential privacy [4] to guarantee the protection of individual data values.

Creating synthetic databases that protect individual records is in fact a common goal in privacy research. Existing results show that if such a synthetic instance accurately preserves too many properties of the original data, it must violate privacy [3]. Thus, in order to remain private, a synthetic database must be tailored to a relatively small class of properties that it can support accurately, while other properties will necessarily fail to be preserved. The novel question investigated here is whether recent advances in privacy mechanisms can be adapted to preserve, with sufficient accuracy, the properties required for system evaluation.

Our contributions include the following. First, we initiate the study of private workload release by considering the potential privacy threats in an enterprise schema, query workload, and database instance. Second, we propose obfuscating transformations for the schema and workload, and we adapt state-of-the-art privacy mechanisms to the release of key statistics about the instance. Third, we carry out preliminary experiments showing that the level of noise required to satisfy privacy standards is acceptable for system evaluation tasks, particularly for large databases.

2 A framework for safe workload release

In our framework, the *owner* of an enterprise database is interested in outsourcing performance analysis tasks but must ensure that sensitive information is protected. The owner's data consists of S, a relational schema including data types, domains, and foreign key constraints; a database instance D conforming to S; and a query workload W, consisting of a collection of SQL queries over S.

The *analyst* would ideally like to acquire the entire collection (S, D, W) to carry out performance analysis. Due to privacy concerns, we release only a transformed version of these objects. Our approach is to obfuscate the schema by transforming S into an isomorphic schema S', and to transform W into W' by re-expressing queries in W in terms of the new schema S' and translating constants in a manner described below. We do not release a surrogate dataset in place of D. Instead, we first compute a set of statistics ST on D, and then we transform it into ST' using a differentially private algorithm. These statistics can be seen as the parameters of a simple statistical model of D. Given that these statistics are computed using an algorithm that satisfies differential privacy, they can be safely released to the analyst.

The analyst, in possession of S', W', and ST', can generate a synthetic database instance consistent with the schema and statistics. There are typically many instances consistent with ST', 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.

Figure 1 provides an overview of the roles of the owner and analyst, the process of translation for S, D, and W, and the generation of synthetic database instances. In the following sections we explain each transformation processes in detail.

2.1 Schema and Domain Translation: $S \rightarrow S'$

The schema S is transformed into S' by obfuscating table and attribute names, and by transforming (or normalizing) the domains of the attributes. To transform S into S', we map each relation in S to a new relation in S'. For each relation in S, we map each attribute to a new relation in the corresponding table. We denote this one-to-one mapping ϕ. The result is a schema S' that is isomorphic to S, but with attribute and table names replaced with canonical values. We also preserve key and foreign key constraints, mapping them consistently to S'.

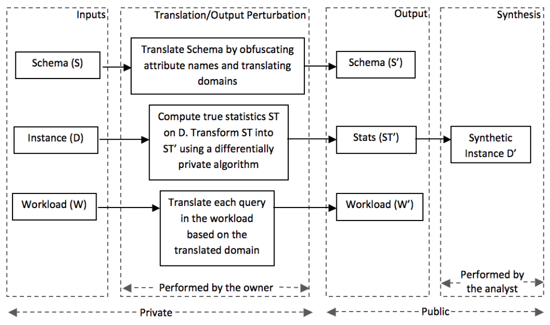

Inputs
Translation/Output Perturbation
Output
Synthesis
Schema (S)
Translate Schema by obfuscating attribute names and translating domains
Schema (S')
Instance (D)
Compute true statistics ST on D. Transform ST into ST' using a differentially private algorithm
Stats (ST')
Synthetic Instance D'
Workload (W)
Translate each query in the workload based on the translated domain
Workload (W')
Performed by the owner
Performed by the analyst
Private
Public

Fig. 1. The schema S, database instance D, and workload W are translated by the *owner* into schema S', differentially-private statistics ST', and workload W'. Using ST', the *analyst* can create one or more synthetic instances.

If data types and detailed domain information reveal too much about the schema, they may be obfuscated. For example, if R_1.salary is an attribute with a domain consisting of integers between 25,000 and 200,000, we may transform and generalize the domain to be integers between 0 and 500,000.

It is possible to consider more complex mappings between S and a new schema, including those that insert/remove columns or tables. This may offer better protection of the original schema, but at a cost of fidelity for the physical properties of the synthetic instances. We leave more complex mappings as a possible direction for future work.

2.2 Workload Translation: $W \rightarrow W'$

We apply the mapping ϕ to the workload of queries expressed on S to get a workload of queries expressed on S'. We assume it is acceptable to release structurally equivalent queries over the translated schema. However, constants appearing in queries are often closely related to actual data values contained in the database, and as a result, could be sensitive. We map query constants to the translated domain, thereby obfuscating them, but preserving the basic relationship of constants to the transformed domain.

Example 1. We use the TPC-H schema in examples and later experiments. Suppose we have the following query in the true workload: SELECT * FROM Lineitem, Orders WHERE L_shipdate $>=$ c_1 AND L_shipdate $<$ c_2. If the true domain for shipdate consists of dates between (d_{min}, d_{max}) and the mapped domain is (d'_{min}, d'_{max}), then we map c_1 to $c'_1 = d'_{min} + (c_1 - d_{min})$ and c_2 to $c'_2 = c'_1 + (c_2 - c_1)$.

2.3 Private Database Statistics

The greatest privacy risk of releasing database workloads is the sensitivity of the contents of the database itself. So while we settle for simply obfuscating schemas and queries, we take a more rigorous approach to protecting data values. We adapt recent privacy techniques to the task of computing a set of statistics describing salient properties of the data owner's instance, D. Because these statistics are computed to satisfy the rigorous standards of differential privacy, the owner can be confident that the data values are protected, subject to appropriate choice of privacy parameters. The analyst may then use the statistics to generate database instances D', similar to D.

The choice of statistics computed from D is a crucial aspect of our framework, and will vary depending on the schema, the workload, and the requirements of the intended tasks to be performed by the analyst. Informally, the more detailed the statistics that are computed, the greater the distortion must be to maintain privacy. In addition, because we ask for multiple statistics, we need to carefully determine the relative importance of accuracy of each statistic to utilize our privacy budget effectively.

Statistical models of database instances We use statistics to model properties of columns, or sets of columns, in relational tables. Our most expressive models of database are joint histograms reflecting the distribution of values across multiple columns. However, depending on how columns are used in workload queries and the properties of the original table that need to be preserved, the owner may choose more or less descriptive statistics for columns. For each table we begin by estimating (privately) the number of records in the table. We then use one of the following models for each non-key column.

Null Model of B_i The null model for a column reflects no additional information beyond what is known about the domain for that column. Modeling a column using a null model does little more than ensure that it will occupy the proper space on disk.

Distinct Values Model of B_i This model has a single statistic: the number of distinct values in column B_i.

Histogram Model of B_i This model divides the domain of B_i into k bins and records the number of tuples in the relation having values in each bin.

Joint Histogram Model of $B_i, B_{i+1}, \dots B_{i+j}$ This model divides each attribute domain into $k_1, \dots k_j$ buckets and reports a joint histogram, i.e. the number of tuples contained in each cell defined by the buckets.

Foreign Key Model of B_i This model is valid for attributes B_i which are foreign keys referencing another relation. The model reports the frequency with which keys in the referenced relation occur in the column.

Prior to workload release, the owner must decide how to assign models to each of the columns in each relation in the database. The statistics for any set of models can be computed privately (as described in Sec. 3). So the selection of column models is primarily motivated by utility considerations. Models must be selected to preserve properties of the database relevant to the query workload and the performance tuning tasks of interest. More descriptive models may better reflect the properties of the database, but the privacy cost associated with those models may ultimately not yield better utility for system evaluation tasks.

Example 2. Suppose we have a join query on orders and lineitem, with a predicate on l_shipdate. We can assign models as follows: histogram model to l_shipdate, foreign key model to l_orderkey, primary key model to primary keys, null model for other attributes. This preserves the primary properties of the database relevant to the query.

2.4 Database Synthesis using Statistics

We use ST' to refer to the collection of statistics for each model associated with D. Given the statistics it is possible to synthesize one or more synthetic instances D'. When properties of a database are not recorded in ST' we simply assume independence by default. For example, if columns B_i and B_j are each modeled using distinct histograms, then we generate values for each column according to the given distributions, but we have no information about how these column values are paired together, so we assume independence. If cross-column correlations are to be preserved in the released database, then a joint histogram model must be used.

Under these assumptions, ST' defines a space of possible database instances consistent with the statistics, and the process of database synthesis consists in sampling from this space of instances. We have implemented a sampling procedure supporting each of the model types above, except for the joint histogram model. In Section 4 we use this preliminary implementation to test the synthetic database instances generated from various models of the TPC-H benchmark database.

Suppose the original table has N rows and a column B_i with domain $[min, max]$. Suppose also the mapped column in the corresponding synthetic table is called B'_i, and the corresponding mapped domain is $[min', max']$. The column B'_i is populated using one of the following generators:

1. **Null generator**: In this case, B'_i is simply generated by picking N random numbers from the range $[min', max']$.
2. **Distinct generator**: Let D be the total number of distinct values in B_i. This generator divides $[min', max']$ into $D - 1$ subintervals and picks D distinct values as the subinterval boundaries, it then randomly picks one of these D values N times to generate B'_i.
3. **Histogram generator**: Suppose the histogram on B_i is (x_1,c_2),...,(x_m,c_m), where x_i is a bucket and c_i is its bucket count. Buckets are mapped to the translated domain, say to x'_1,..., x'_m. To generate B'_i, the generator picks c_i values at random from bucket x'_i.
4. **Foreign Key generator**: Suppose B_i is a foreign key that references Z_i in another table. Let the corresponding synthetic columns be B'_i and Z'_i. Suppose the distinct value histogram on B_i is (y_1,c_1), ...,(y_p, c_p) where y_i is a distinct value in B_i and c_i is its frequency. The foreign key generator generates B'_i consistent with the histogram (y'_1,c_1),....,(y'_p,c_p), where y'_js are p distinct values from the domain of Z'_i.

2.5 Accuracy of synthetic databases

The accuracy of a synthetic database D' is measured in terms of the performance of the workload queries. That is, accuracy is measured as the difference between $cost(W, D)$ and $cost(W', D')$, where *cost* may be one of a number of performance properties. These properties of interest are application dependent, but examples include: query result cardinality, estimated execution metrics (time or IOs), actual execution metrics (time or IOs), qualitative aspects of the query execution plans, and properties of index usage.

For a workload W, we refer to the difference in *cost* between the true database instance D and a synthetic instance D' as *error*, and we distinguish between two contributing sources of error. *Modeling error* results from the fact that the only information the analyst has about the true database instance is that present in the released statistics. Even when the statistics associated with these models are reported without distortion, the resulting model only provides partial information about the true database. Selecting more descriptive models reduces modeling error. *Perturbation error* results from the fact that noise is added to the statistics before releasing them. Thus the space of possible database instance from which the analyst will sample is only approximately representative of the true database. Perturbation error is determined by the privacy parameters, which control the strength of the privacy guarantee required by the owner, as well as the number of statistics computed. In particular, when many related statistics are computed about the database, more distortion must be added to maintain a fixed privacy guarantee.

Note that some of the performance metrics considered above are themselves imprecise. For example, estimates of query execution time or IOs typically differ from actual measures. And actual execution times vary based on system state and load. Our hope is to achieve rates of modeling and perturbation error that are small, *relative to the imprecision or variation inherent in common performance measures.*

3 Privacy Preserving Methods

In this section we describe the privacy guarantees provided by our framework along with the privacy mechanisms used to compute database statistics. We use the standard of *differential privacy* [4], which offers participants in a dataset an assurance that information released about the dataset is virtually indistinguishable whether or not their personal data is included. It protects against powerful adversaries, and offers precise, quantifiable accuracy guarantees.

Differential privacy is achieved by randomizing the answers to queries over a sensitive database. To adapt the techniques of differential privacy to our context, we can view the statistics ST as a set of aggregate queries over D (e.g. predicate counting queries for histograms, or count-distinct queries for the distinct generator). Using a differentially private algorithm to compute these queries will result in noisy answers, which constitute the private statistics ST'.

We use (ϵ, δ)-differential privacy [10], sometimes called approximate differential privacy, which places a bound (controlled by ϵ) on the difference in the probability of any query answer on neighboring databases, but allows that bound to be violated with small probability (controlled by δ). The definition of differential privacy relies on the concept of *neighboring* databases, which are two database instances that differ by exactly one tuple, denoted $nbrs(I, I')$.

Definition 1 (Approximate Differential Privacy). *A randomized algorithm $\mathcal{K}$ is (ϵ, δ)-differentially private if for any instances I, I' such that* $\mathrm{nbrs}(I, I')$*, and any subset of outputs $S \subseteq Range(\mathcal{K})$, the following holds:*

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

Limitations of the guarantee While differential privacy offers one of the strongest guarantees considered by the privacy community, it is important to consider the implications of adapting differential privacy to our objective of safe workload release. Differential privacy is designed to protect the sensitive information of individuals. When each individual's information is contained in a single tuple, an individual can be confident in allowing their data to be included in the dataset because the answers released are guaranteed to be virtually indistinguishable from those released in the absence of their data. In most settings, the hope is that this individual guarantee can be maintained while aggregate properties of the database can be released. Indeed, a statistic like the total number of tuples in the database does not depend much on any one person's information, and will typically be estimated very accurately under differential privacy.

As noted in the previous discussion, the privacy concerns in workload release may go beyond the protection of individual tuples. So in some settings it may not be acceptable to accurately release aggregate properties of a database. For example, if the size of a table in an enterprise database will reveal the number of sales completed by the enterprise, and this fact is sensitive, then protecting single tuples is not an adequate privacy standard. Differential privacy can easily be adapted to offer a form of group privacy, in which any set of k tuples are protected. This is achieved with exactly the same techniques, but requires increasing the noise added to query answers. We adopt this solution, but recognize that for some applications even this guarantee may not be satisfactory. In such cases, it may not be feasible to accomplish safe workload release.

3.1 Differentially-private algorithms

In order to satisfy these definitions, the noise added to a query answer must be calibrated to the *sensitivity* of the query, a static property of a query which reflects the worst case impact the addition or deletion of one tuple can have on the output. Since we will work with sets of queries describing the statistics in ST, we define sensitivity for a vector of aggregation queries.

Definition 2 (Sensitivity). *Let* $\mathbf{Q}$ *represent a vector of aggregation queries. The* L_2 *sensitivity of* $\mathbf{Q}$*, denoted* $||\mathbf{Q}||_2$*, is* $||\mathbf{Q}||_2 = \max_{\{I,I' | \text{nbrs}(I,I')\}} ||\mathbf{Q}(I) - \mathbf{Q}(I')||_2$

For arbitrary functions, computing the sensitivity may be undecidable. However for the queries underlying all statistics mentioned Sec. 2.4, the sensitivity is easily computed.

Example 3. Suppose $\mathbf{Q}$ is a vector of k disjoint range-count queries representing a histogram over an integer attribute with domain [0..100). For example, we can write $\mathbf{Q}$ as $(q_1 \ldots q_k)$ where each q_i counts the number of tuples in one of a disjoint set of ranges over the domain of a single attribute. Then the addition or deletion of one tuple in I will change exactly one component of $\mathbf{Q}(I)$ by exactly one. Therefore, $||\mathbf{Q}||_2 = 1$.

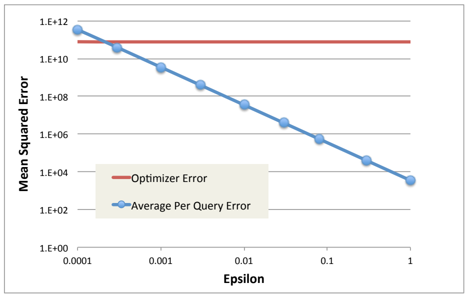
Approximate differential privacy can be achieved by adding Gaussian noise calibrated to the L_2 sensitivity of the queries. The following proposition defines an algorithm for achieving approximate differential privacy for any vector of aggregate queries:

Proposition 1 (Gaussian mechanism). *Given a vector of aggregate queries* $\mathbf{Q}$*, of length* k*, the randomized algorithm* $\mathcal{G}$ *that outputs the following vector is* (ϵ, δ)*-differentially private:*

$$\mathcal{G}(\mathbf{Q}, I) = \mathbf{Q}(I) + \mathit{Normal}(\frac{||\mathbf{Q}||_2}{\epsilon}\sqrt{2\ln(2/\delta)})^k$$

The Gaussian mechanism adds k independent samples to the true answer to $\mathbf{Q}$, where the samples are drawn from a Gaussian distribution scaled to the sensitivity of $\mathbf{Q}$. This differentially-private mechanism is sufficient for computing each of the statistics we use to model databases. To do so, we would construct a single query vector containing all statistics for the chosen models, compute the sensitivity, and add noise accordingly. Notice that adding additional statistics that increase the sensitivity will increase the noise added to *each* of the statistics. Although this mechanism is sufficient for the modeling tasks discussed in the previous section, the accuracy of our statistics can be improved by two recently-proposed techniques [8, 6] which we describe briefly next.

Differentially private histograms The Gaussian mechanism is sufficient for ensuring differential privacy, but the error rates achieved by the mechanism are not optimal, particularly when it is applied to multiple queries. The recently proposed matrix mechanism [8] allows for significantly lower error for workloads consisting of sets of counting queries, including collections of low-order marginals and multi-dimensional histograms. The matrix mechanism exploits correlation in these sets of queries and adds a more complex noise distribution which follows correlation in the queries. For fixed ϵ, the mechanism can reduce error rates from $O(n^2)$ to $O(log^3 n)$ for set of range queries, where n is the size of the domain. We use these techniques to compute the private statistics for histogram models.



(a) Error, 3-dimensional histogram

Query	Optimizer Estimate	Model	Private Model
Q1	1.000083	1.000086	1.000086
Q2	1.010152	1.000087	0.999967
Q3	4.191747	1.000742	1.000381
Q4	1.734864	0.998955	1.004799
Q5	1.466320	0.200764	0.199519

(b) Error, synthesized 3-table database

Fig. 2. Error in query result sizes for range queries over the TPC-H schema.

Differentially private frequencies Another recent technique [6] has improved the accuracy of differentially-private estimates of frequency distributions. This technique can be used to model key/foreign-key relationships between tables. If column A is a foreign key referencing table R, then in order to model the correct join frequencies, we need to gather statistics about the frequency of occurrence of each value in A. In our setting, we compute these frequencies and then choose randomly from the set of canonical key values generated for the synthesized table R. The naive method for estimating such frequencies results in $O(m)$ total expected error, where m is the size of the database. The improved technique employing post-processing of the noisy frequencies [6] can reduce this error to $O(dlog^3m)$ where d depends on the number of distinct frequencies in the sequence, which is likely to be low in practice.

4 Performance Analysis

To assess the feasibility of accurate outsourced system evaluation we carried out two preliminary experiments. In both cases we use the TPC-H benchmark database, scale factor 1, and Postgres 8.1. We focused on query result cardinality as a performance metric. This case would be interesting for a database researcher who wishes to evaluate selectivity estimation techniques.

In the first experiment, we investigate the overall error achievable when modeling a single relation using a multidimensional histogram. Using the privacy method for histogram queries described in Sec 3.1 we analytically compute the expected error in estimating the output cardinality of three dimensional range queries. These error rates vary with choices of the privacy parameters δ and ϵ. We fixed δ at 10^{-5}, which means that the probability of violating the bound on disclosure is quite low. Figure 2(a) (in log-log scale) shows the relationship of error as a function of ϵ. To put these error rates in context, we empirically calculated the average error of the Postgres optimizer in estimating the output cardinality of similar range queries. To do this, we generated 500 random range queries each imposing a range condition on three attributes of the Lineitem table from the TPC-H schema. Our conclusion is that the error of the privacy mechanism is acceptable, even for conservative privacy settings.

In the second experiment, we applied our synthetic data generators to a subset of the TPC-H schema: the Orders, Lineitem, and Customer tables. Due to lack of space we present only selected results. We considered queries consisting of the natural join of the three included tables, along with one or more selection queries on individual attributes from the tables. In the table, $Q1$ is the natural join alone, $Q2, Q3, Q4$ have one range condition each, on Orders, Lineitem, and Customer, respectively. $Q5$ combines three range conditions on the tables. We chose a model designed to support these queries. It uses the null model for all attributes not used in the queries, foreign key models for all foreign keys, and one-dimensional histograms on attributes appearing in any WHERE clause.

We evaluated these queries on the original TPC-H tables, synthetic tables generated using the true statistics (the **Model**), and synthetic tables generated using private statistics (the **Private Model**). The table in Fig. 2(b) reports, for each query, the result size estimated by the optimizer, computed on the Model database, and computed on the Private Model database. Result sizes are reported as a ratio of the true result size of the query on the original database.

Overall, model and perturbation error is low for queries with one selection condition, suggesting that key-foreign key joins are modeled accurately, even under the privacy condition. Model error is high for multi-attribute range queries, reflecting a limitation in our foreign key generator: it models the “out-degree” of records, but not correlation in foreign key references to attributes of the referenced table.

5 Related Work

The work closest to our own is a framework for private database synthesis proposed by Wu et al. [13]. Similar to our approach, they release a set of rules representing database constraints along with statistics, allowing an external party to generate a synthetic database. Wu et al. have also extended this framework with a more expressive statistical model [12]. Their privacy condition, however, is significantly different from ours. They protect privacy by allowing the data owner to specify a set of sensitive properties and then transforming the rules and statistics to avoid disclosure of the properties. Differential privacy offers a more rigorous privacy guarantee and the opportunity to quantify the error of the output.

In the absence of privacy concerns, generating realistic synthetic relational data has received considerable attention in the research community. Some researchers have focused on the underlying data distributions and characteristics of database instances [2, 5, 7, 11]. Bruno et al. [2] propose a specification language that can be used to select existing iterators, or define new iterators based on data distributions, specify inter-table correlations, and they formalize synthetic database generation. Gray et al. [5] have considered how to design very efficient generators for large databases, exploiting parallel computation. Houkjaer et al. [7] propose a graph model that encodes underlying distributions and parameters, which can be configured by the user. The Up-Sizer system [11] is intended to preserve statistical properties of a source database while allowing the generation of larger instances for the realistic investigation of the impacts of scale-up on a system. More recently, query-aware mechanisms for synthetic database generation have been proposed. These techniques can produce workloads satisfying user-provided constraints on the cardinality of intermediate operators for specified queries [1, 9].

6 Conclusion

We propose a framework for supporting secure outsourced system evaluation through the private synthesis of database instances and the translation of workloads. The results of our preliminary performance evaluation show that the levels of error due to the privacy mechanism are acceptable, suggesting that accurate database synthesis should be possible. Remaining challenges include extending to full schemas and larger workloads, as well as modeling cross-table correlations more accurately.

References

1. C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: Generating query-aware test databases. In *SIGMOD*, 2007.
2. N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, 2005.
3. I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.
4. C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
5. J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. *SIGMOD Record*, 23, 1994.
6. M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially-private histograms through consistency. In *VLDB*, 2010.
7. K. Houkjaer, K. Torp, and R. Wind. Simple and realistic data generation. In *Conference on Very Large Databases*, pages 1243–1246, 2006.
8. C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, 2010.
9. E. Lo, N. Cheng, and W.-K. Hon. Generating databases for query workloads. *Proc. VLDB Endow.*, 3:848–859, September 2010.
10. F. McSherry and I. Mironov. Differentially Private Recommender Systems : Building Privacy into the Netflix Prize Contenders. In *SIGKDD*, 2009.
11. 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.
12. X. Wu, Y. Wang, S. Guo, and Y. Zheng. Privacy preserving database generation for database application testing. *Fundamenta Informaticae*, 78(4):595–612, 2007.
13. X. Wu, Y. Wang, and Y. Zheng. Privacy preserving database application testing. In *Workshop on Privacy in the Electronic Society (WPES)*, pages 118–128, 2003.