

# Privacy Preserving Database Application Testing

Xintao Wu  
CS Department, University of  
North Carolina at Charlotte  
xwu@uncc.edu

Yongge Wang  
SIS Department, University of  
North Carolina at Charlotte  
yonwang@uncc.edu

Yuliang Zheng  
SIS Department, University of  
North Carolina at Charlotte  
yzheng@uncc.edu

## ABSTRACT

Traditionally, application software developers carry out their tests on their own *local development* databases. However, such local databases usually have only a small number of sample data and hence cannot simulate satisfactorily a live environment, especially in terms of performance and scalability testing. On the other hand, the idea of testing applications over *live production* databases is increasingly problematic in most situations primarily due to the fact that such use of live production databases has the potential to expose sensitive data to an unauthorized tester and to incorrectly update information in the underlying database. In this paper, we investigate techniques to generate *mock* databases for application software testing without revealing any confidential information from the live production databases. Specifically, we will design mechanisms to create the deterministic rule set  $\mathcal{R}$ , non-deterministic rule set  $\mathcal{NR}$ , and statistic data set  $\mathcal{S}$  for a live production database. We will then build a security Analyzer which will process the triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  together with security requirements (security policy) and output a new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ . The security Analyzer will guarantee that no confidential information could be inferred from the new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ . The mock database generated from this new triplet can simulate the live environment for testing purpose, while maintaining the privacy of data in the original database.

## Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging—*testing tools*; H.1.1 [Models and Principles]: Systems and Information Theory—*information theory*; H.2.8 [Database Management]: Database Applications—*statistical databases*

## General Terms

Algorithms, performance, security, theory

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WPES'03, October 30, 2003, Washington, DC, USA.  
Copyright 2003 ACM 1-58113-776-1/03/0010 ...\$5.00.

## Keywords

Database application testing, privacy, indistinguishability

## 1. INTRODUCTION

The amount of sensitive information about citizens accumulated in the databases of government agencies and private organizations, such as Social Security Administration, banks, and health care providers, has been increasing steadily in the past decades. While it has long been realized that there is a need to protect the information both in storage and transition, it has recently become apparent that the information needs to be properly guarded from unauthorized disclosure during the process of testing newly developed applications that employ the databases.

Testing is essential for database applications to function correctly and with acceptable performance when deployed. Currently, two approaches dominate database application testing. With the first approach, application developers carry out their tests on their own *local development* databases. Obviously this approach can not fulfill the requirements of all the testing phases, especially those pertinent to performance and scalability, due to the limitation of relatively small size of data and test cases. Furthermore, the data in *local development databases* may not be accurate or close to real data. With the second approach, new applications are tested over *live production* databases. This approach cannot be applied in most situations due to the high risks of disclosure and incorrect updating of confidential information.

In this paper, we investigate a novel method for database application testing by generating *mock* databases based on some a-priori knowledge about the current *production* databases without revealing any confidential information. In the *mock* databases, we will have almost identical structures (table schema description, view, index etc.) and statistics to *production* databases which contain live data and are used by real applications. The generated data in *mock* databases will be able to help organizations to arrive at a close estimate of the performance of a database application, which can be significant for organizations which need to quantify the potential gain in performance.

The data in our mock databases will have three characteristics: *valid*, *resembling real data*, *privacy preserving*. To be valid, the generated data need to satisfy almost all the constraints and business rules underlying the live data. To resemble real data, the generated data need to have similar statistical distributions or patterns to the live data in almost all possible levels (table, column, etc.), at least for the

purpose of database application testing. In testing the performance of database applications, it will be imperative that the data is realistic (valid and resembling real data) since the statistical nature of the data determines query performance. To be privacy preserving, the generated data should not disclose any confidential information that the database owner would not want to reveal. There are several kinds of confidential information that a database owner would like to protect. An incomplete list could include: the existence of some fields in a table, some statistical data about the live database, some deterministic or non-deterministic business rules or constraints of the live database, and users’ records in the tables. The mock database will be constructed in such a way that none of these (database owner specified) confidential information will be contained in it or can be inferred from it. Since we will generate the mock database from random values, the most important part of our techniques will be concentrated on the protection of indirect information leakage from the mock database.

Our approach is more feasible than other approaches. First, our approach is to generate data specifically for the purpose of testing and to run the tests in an isolated environment. The databases can possibly be shared by all developers so they can run their applications and see how it work with either realistic amounts or any amounts of data, rather than a handful of records in a local *development* database. Second, the a-priori knowledge required is generally available from detailed entity-relation diagram (ER) or schema definitions in data definition language (DDL) with complex data integrity rules as well as statistical information as an organization implementing a complex database application usually has clear understanding of the nature of the data on which the system will operate. Third, our approach can achieve better *controllability, observability, and privacy*. Using synthetic data, we can put a database system into the desired state before executing test cases (controllability) and observe its state after the execution of the test cases (observability). Two further potential threats, namely direct disclosure of individual data and indirect interpretation from the disclosed data to confidential data, are prevented by using synthetic datasets. A further advantage over other approaches is that there is no need in our approach to access the confidential live data. A formidable challenge that will be addressed in this paper is to minimize indirect confidential information leakage (e.g., the inference of some deterministic or non-deterministic business rules of the original database).

Theoretically, there are two types of indirect confidential information leakage [26]: re-identification disclosure and prediction disclosure. Re-identification disclosure occurs if an attacker is able to deduce the values of a sensitive attribute for a target individual after this individual has been re-identified (see, e.g., [28]). Prediction disclosure occurs if the data enable the attacker to predict the value of a sensitive attribute for some target individual with some degree of confidence. In this project, we will study mechanisms to automatically delete or modify these statistical data which may be used for re-identification or prediction.

Strong cryptographic techniques such as provable security, indistinguishability, and perfect privacy [13, 14, 36] will be extensively used in the design of mock database. A central notion in modern cryptography is that of “effective similarity” introduced by Goldwasser, Micali, and Yao [14, 36].

The underlying idea is that we do not care whether objects are equal, all we care is whether or not a difference between the objects can be observed by an efficient computing device. This notion has been used to define the simulation paradigm on which the notions of provable security (such as zero-knowledge protocols) and perfect privacy are based (see, e.g., [12]). The simulation paradigm will be used to define the requirements for our mock database security and effectiveness.

There are two indistinguishable properties that we need to consider. In order to achieve the best simulation environment for the testing, we will construct the mock database such that for an honest database application test, the mock database is indistinguishable from the live database (note that for a malicious application tester which tries to get some confidential information from the database, this mock database could be distinguishable from the live database). Secondly, in order to keep perfect privacy of the confidential information in the live database, the knowledge about the confidential information that a database application tester learns from the mock database is indistinguishable from the knowledge that he could have already learned without access to the mock database. In other words, the database application tester gains nothing substantial about any confidential information on the live database from the mock database by deviating from the behavior exhibited by an honest database application tester.

We believe that this work represents the first application of using synthetic data generation for privacy preserving database applications testing. The rest of the paper is organized as follows. In Section 2 we review related work. We present our method in Section 3. In Section 4 we draw conclusions and describe directions for future work.

## 2. COMPARISON WITH RELATED WORKS

In practice, it is often necessary for a database software vendor to test their software on a live commercial database before selling or integrating their package to the database owner. The testing of database applications can be classified as: functional testing, performance testing (load and stress, scalability), environment and compatibility testing, and usability testing. In this paper, we focus on performance testing which identifies current bottlenecks in application and verifies whether it meets or exceeds key performance measures. The crucial point here is how we can design the testing environment so that all functions of the software package are tested while no confidential information of the real database is leaked.

One approach to testing database applications is to simply use live data. Wiederhold et al., in [33, 34], investigated how to protect inappropriate release of data from realistic databases. Figure 1 shows the architecture of modified live testing by introducing a security filter to protect privacy. The security filter lies between the tested applications and *production* databases. The security filter regulates access to database information by screening queries to databases and contents of results to applications. The security filter has one filter database, which stores the table description of production tables, the rules table (containing the policy rules that govern query and result screening), and the audit table (a record of all transactions, including date, time,

queries, results, and possible rule violation statements). For a read-only query, the security filter will forward the query to real databases, combine the results with local records if necessary (e.g., the query involves previous-updated values), and transform query results (by mocking or distorting sensitive information based on predefined rules) to applications. For write queries, the security filter will update information locally.

This approach has several disadvantages. The live data may not reflect a sufficiently wide variety or possible situations that could occur. That is, testing would be limited to, at best, situations that could occur given the current DB state. It is also difficult to identify appropriate user inputs to exercise them and to determine appropriate user outputs. More importantly, testing with data that is currently in use may be dangerous since running tests may corrupt the database so that it no longer accurately reflects the real world. This may be difficult, particularly if the changes are extensive or if other real modifications to the database are being performed concurrently with testing.

It is also possible to duplicate the live production database and build a security filter to protect database privacy. However, the security filter may reduce the effect of performance testing since the filter could be complicated and slow.

Since using live data is problematic, our approach is to generate data specifically for the purpose of testing and to run the tests in an isolated environment. Although there has been some prior investigations into data generation by a limited number of researchers, the tools currently available [24, 22, 29, 16] for synthetic data generation are built either for testing data mining algorithms only and thus limited to small data mining domains or for assessing the performance of database management systems, rather than testing database applications. In addition, they lack the required flexibility to produce more realistic data needed for performance testing or benchmarking databases in general.

Our approach addresses three problems: controllability, observability, privacy preserving. The controllability and observability are very important for database application testing as the input and output spaces of testing program include the database states as well as the explicit input and output parameters of the application. The requirements of complete application testing pose threats to the privacy of underlying data. Some of these can be directly attributed to the direct disclosure of some individual data. Others can be attributed more to the interpretation, application and actions taken from the disclosed data. These threats are expected to increasingly raise concerns. The current state of the art is that there has been little work dedicated to privacy preserving application testing, although some approaches from the field of statistical databases could potentially be extended and adapted to develop new techniques to balance the rights to privacy and the needs for accessing the underlying data. The field of statistical databases [1, 9, 8] has developed methods to prevent the disclosure of confidential individual data while satisfying requests for aggregate information (sum, count, average, maximum, minimum,  $p$ th percentile, etc.). Experience from this field indicates that removing identifiers such as names, addresses, telephone numbers and social security numbers satisfies only a minimum requirement for privacy and hence is not adequate. Re-identification based on remaining fields may still be possible and removing identifiers is considered the weak-

est approach, and should never be used on its own. The proposed advanced techniques can be broadly classified into query restriction and data perturbation [2]. The query restriction family includes restricting the size of query result, controlling the overlap amongst successive queries, keeping audit trail of all answered queries and constantly checking for possible compromise, suppression of data cells of small size, and clustering entities into mutually exclusive atomic populations. The perturbation family includes swapping values between records, replacing the original database by a sample from the same distribution, adding noise to the values in the databases, adding noise to the results of a query and sampling the result of a query [1].

Recently, Malin, Sweeney, and Newton [28] have presented several algorithms for learning the identities of individuals from the trails of seemingly anonymous information they leave behind. Since our mock database will be generated from random values by using some modified rules and statistical dataset, these attacks are not applicable to our approach.

We should also point out that the privacy consideration in the current literature for statistical database is not enough for most environments. In most statistical database literatures, the privacy concerned is about the re-identification of some specific entries in the database. For example, when Dinur and Nissim [8] examine the tradeoff between privacy and usability of statistical databases, they have the following definition of privacy (we present their idea informally, the reader is referred to [8] for their original definition): The scenario is modeled by a two-phase adversary. In the first phase, the adversary is allowed to adaptively query the database. At the end of this phase, the adversary commits to a challenge—an index  $i$  of the entries in the database it intends to guess. In the second phase, all the database entries except the  $i$ th entry are revealed to the adversary. The adversary succeeds if it outputs the  $i$ th entry correctly. In most databases such as financial databases, some rules and statistical data about the live production database is also considered confidential information. As we have seen, statistical database approach does not try to protect this kind of confidential information at all. In our mock database approach, we will develop techniques to protect all confidential information that the database owner specifies.

Also related to our research is private information retrieval and privacy preserving data mining [2]. The theoretical work of private information retrieval [5] enables users to obtain information from databases while keeping their queries secret from the database managers. Symmetrically private information retrieval [6] addresses the database's privacy as well by adding the requirement that the user, on the other hand, cannot obtain any additional information about the database in a single query except for a single physical value. These schemes are generally theoretical and have high complexity. The objective of privacy-preserving data mining [2, 3, 10] is to prevent the disclosure of confidential individual values while preserving general patterns and rules. As most patterns rely on the statistics of some attributes which have a high impact on the patterns and rules to be discovered. The techniques used there include data distortion approach (for decision tree [2], association rules [11, 25]), cryptographic approach (decision tree over distributed data [21], k-means clustering over vertically partitioned data [32], association rule over horizontally partitioned data [17] and

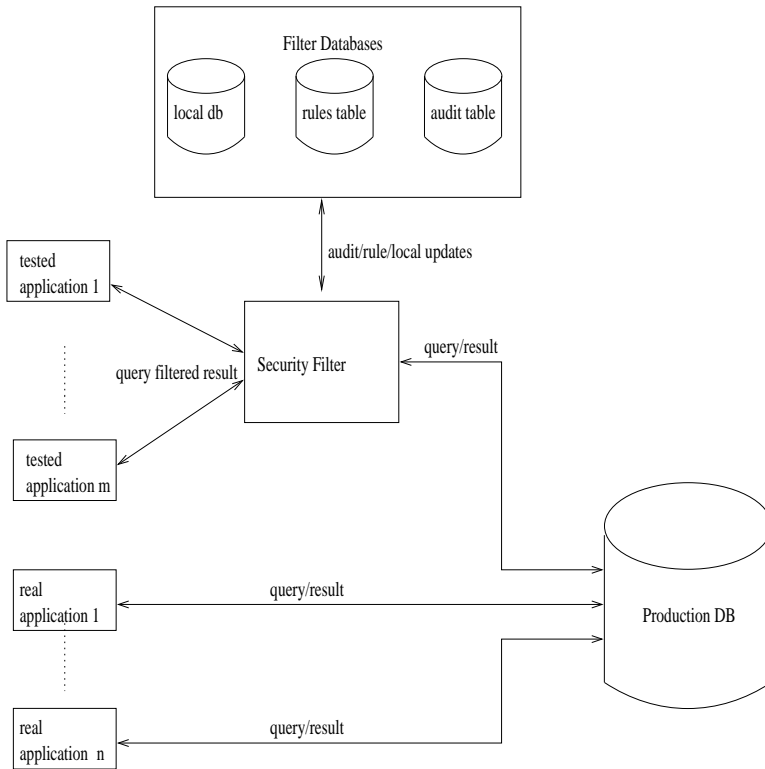


Figure 1: Architecture of live testing approach

vertically partitioned data [31]). The idea of data distortion approach is that the distorted data, together with the distribution of the random data used to distort the data, can be used to generate an approximation to the original data values while the distorted data does not reveal private information, and thus is *safe* to use for mining. The idea of cryptographic approach is to use multi-party computation technique [35] compute the required aggregate information, without each source revealing sensitive information.

### 3. OUR METHOD

When a database application software vendor wants to test the performance of its software, it can generate a mock database by simply synthesizing random data into the database and then test its software on it. However, this approach will not produce a close estimate on the software performance. A much better approach that we will take is as follows. In order to make the mock database look closely to the live production database, we can extract some rules and statistical data from the live database and then synthesize random data into the mock database according to these rules. In particular, we extract the triplet set  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  from the live database in such a way that it will guarantee the generated synthetic data in mock databases valid and close looking to real data. We use  $\mathcal{R}$ ,  $\mathcal{NR}$ , and  $\mathcal{S}$  to denote deterministic rule set, non-deterministic rule set, and statistics set for a database respectively. The deterministic rule set,  $\mathcal{R}$ , includes deterministic rules (e.g., domain constraint, uniqueness constraint, referential integrity constraint, functional dependencies, and semantic integrity constraint etc.) while non-deterministic rule set,  $\mathcal{NR}$ , contains

non-deterministic information (e.g., association, correlation, pattern etc.). Statistics set,  $\mathcal{S}$ , contains the *statistics* about the database instance (e.g., the cardinality of a table, value sets or ranges of each column, the frequencies of column values or statistical distributions etc.). Figure 2 shows the architecture of our rule based mock database generation system.

There are two major problems that need to be addressed: 1) some rules in the triplet set  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  may be inaccurate or conflict with another rule due to errors in design or in domain knowledge; 2) some rules may contain sensitive or confidential information about the database. Thus the rule Analyzer component will be applied here to derive an accurate and privacy preserving  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  by hiding or replacing some rules (or statistical data). The information contained in the triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  is the same as the information contained in the mock database. Thus it is sufficient to guarantee that the triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  does achieve the three characteristics: valid, resembling (to the original triplet), and privacy preserving (i.e., no confidential information could be inferred from this triplet). We give the formal definitions of similarity and privacy-preserving in Section 3.2. In Section 3.1 we focus on how to extract the triplet set  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  from live databases. We present our rule analyzer and mock database generator in Section 3.3 and Section 3.4 respectively. In Section 3.5, we present the implementation issues of our ongoing prototype system.

#### 3.1 Rule-based data specification—generating the triplet $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$

The specification of database testing involves characterizing data values, distributions, and relations. Thus, to

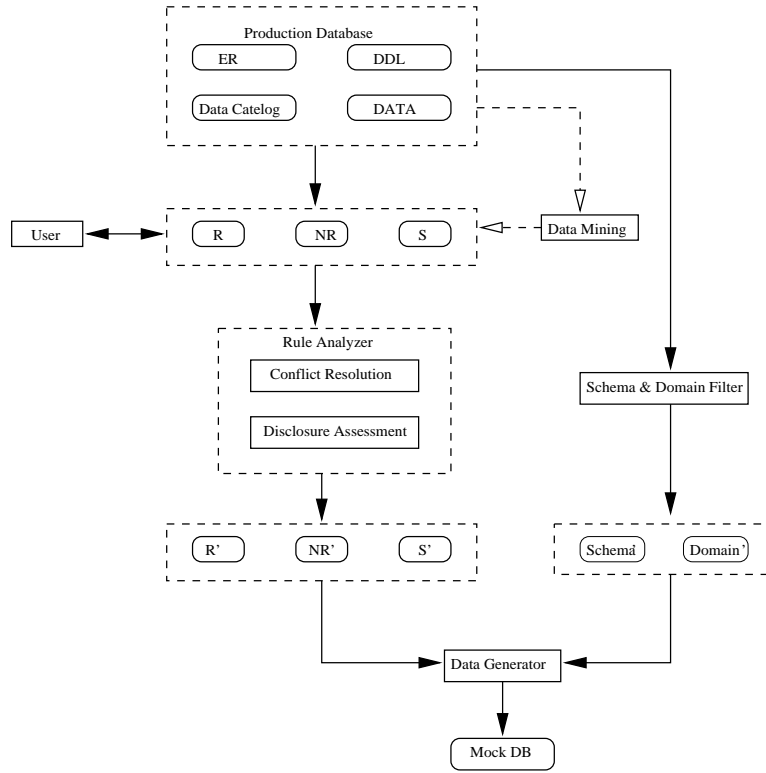


Figure 2: Architecture of rule based data generation system

achieve the goal of generating valid, close looking data, we expect the users to provide knowledge about the values, distribution, relations, and integrity constraints the data embodies.

We assume databases are based on the relational model in our paper. A database in relational model is a collection of one or more relations, where each relation consists of a relation schema and relation instance. A relation schema  $R(A_1, \dots, A_n)$  is a relation name along with a list of attributes, each of which has a name  $A_i$  and a domain  $dom(A_i)$ . A relation instance is a set of tuples, each of which is an element of the Cartesian product  $dom(A_1) \times \dots \times dom(A_n)$ . The integrity constraints restrict the possible values of the database states so as to more accurately reflect the real-world entity that is being modeled. The constraints include domain constraint, uniqueness constraint, referential integrity constraint, functional dependencies, and semantic integrity constraint such as business rules (e.g., current date – date of birth = age, unit price  $\times$  number of units = total expense etc.). It is desirable that the generated data in mock databases also satisfy the constraints.

To derive the above deterministic rule set  $\mathcal{R}$ , we take advantage of the database schema, which describes the domains, the relations, and the constraints the database designer has explicitly specified. The generated data need to satisfy all the constraints and business rules. This process leverages the fact that this information is expressed in a formal language, SQL’s data definition language (DDL) or can be derived from ER diagrams. Some information (function dependencies, correlations, hierarchies etc.) can be derived from database integrity constraints such as foreign keys, check conditions, assertions, and triggers.

In order to ensure that the data is close looking or statistically similar to real data, or at least from the point of view of application testing, we need to have the statistical descriptions,  $\mathcal{S}$ , and non-deterministic rules,  $\mathcal{NR}$ , of real data in *production* databases. These two sets describe the statistical distributions or patterns of underlying data and may affect the size of relations derived as a result of the evaluations of queries the application will need to execute. Hence, they are imperative for the statistical nature of the data that determines the query performance of database application. In this paper we extract the simple statistics directly from database catalog. The complex statistics and non-deterministic information need to either be provided by expert or be discovered by other tools such as data mining or statistical software.

Formally, each rule in  $\mathcal{R}$  and  $\mathcal{NR}$  can be represented as a declarative rule and is generally of the form:

if  $\langle \text{premise} \rangle$  then  $\langle \text{conclusion} \rangle$  [with support  $s$  and confidence  $c$ ].

Both *premise* and *conclusion* are Boolean combinations (that is, an expression using the logical connectives AND, OR, and NOT) of conditions of the form  $\langle \text{expression op expression} \rangle$ , where *op* is one of the comparison operation ( $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $\neq$ ) and *expression* is a column name, a constant, or an (arithmetic or string) expression. The rules may include exact, strong, and probabilistic rules based on the support and confidence. We note here that complex predicates and external function references may be contained in both the condition and action parts of the rule. Anyone

with subject matter expertise will be able to understand the business logic of the data and can develop the appropriate conditions and actions, which will then form the rule set.

The attribute list of relations can be classified as two sets: *independent attribute set* and *dependent attribute set*. These two sets are derived from the deterministic rules of  $\mathcal{R}$ , where each rule may be formed by composite attributes (special handling will be needed for circular chains of referential integrity rules). Obviously, the value for dependent attribute can be derived by the values of independent attribute following some rule.

It is possible that conflicts happen among the rules due to error in design or in domain knowledge. The conflicts can be seen as an optimization or constraint satisfaction problem. We need to resolve the rule conflicts. The advantage of searching through the space of specifications rather than that of data instances is that specifications can then be analyzed to see how the conflict is resolved, and the user can then improve the specification further if needed.

In practice it is possible that the user can not always provide details of rules or statistics. The user only has an approximate and possibly incorrect idea of how the data should be, and can therefore give at most a partial or abstract specification. In this case, we evolve a complete, detailed specification from the user's partial one by filling in the details at random, testing their validity, and repeating this generating and testing process until we obtain a complete specification that most closely matches the original partial one. It may also be true that the user knows little or nothing about the properties of the data he wishes to generate, but can provide a sample of real data. In this case, we use different data mining techniques on the data to essentially get a picture of the properties and relationships of the data, and use this to build a specification which can be used to produce synthetic data with similar properties.

### 3.2 Formal approaches to database close-lookingness and privacy

Two databases  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  are *close-looking* for application performance testing if the application software cannot tell the difference of the two databases in the sense of performance testing. In another word, for a database application software  $\mathcal{M}$ , if we run  $\mathcal{M}$  on both  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  using given test cases  $x$  and get the same performance results, then we say that  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  are close-looking for application performance testing.

The above intuition about the database close-lookingness can be expressed formally in the following definition.

**DEFINITION 3.1.** *Let  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  be two databases,  $x \in \{0, 1\}^n$  is a binary string representing test cases given by users,  $t(x)$  be a time function, and  $\delta(n)$  be a negligible function<sup>1</sup>. We say that  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  are  $(t, \delta)$ -close-looking for application performance testing with regard to an application software  $\mathcal{M}$ , we have*

$$\text{Prob}[|T(\mathcal{M}(\mathcal{DB}_1, x)) - T(\mathcal{M}(\mathcal{DB}_2, x))| \geq t(x)] \leq \delta(n)$$

where  $T(\mathcal{M}(\mathcal{DB}, x))$  is the running time of the application software  $\mathcal{M}$  on the inputs  $\mathcal{DB}$  and  $x$ , and the probability is taken over the choices of the input  $x$  and internal coin tosses of the  $\mathcal{M}$ .

<sup>1</sup>A function  $f(\cdot)$  is called *negligible* if for any polynomial  $p(\cdot)$ , we have  $f(n) \leq \frac{1}{p(n)}$  for large enough  $n$ .

Intuitively, the above definition says that databases  $\mathcal{DB}_1$  and  $\mathcal{DB}_2$  are close-looking for application performance testing if the following statement holds: for an application software  $\mathcal{M}$  and any external input test cases  $x$  (could be empty), the running time of  $\mathcal{M}(\mathcal{DB}_1, x)$  and  $\mathcal{M}(\mathcal{DB}_2, x)$  are approximately the same (with a difference of  $t(n)$ ) with overwhelming probability. We say that an event happens with overwhelming probability if the probability that it does not happen is negligible.

Let  $\mathcal{G}$  be a random process (i.e., a nondeterministic Turing machine) such that, for any consistent triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and any random coin tosses  $r$  (i.e., a random binary sequence),  $\mathcal{G}$  generates a mock database  $\mathcal{DB} = \mathcal{G}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, r)$  which satisfies the deterministic rules, nondeterministic rules, and statistical data in  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ .

**DEFINITION 3.2.** *A random process  $\mathcal{G}$  is called a  $(t, \delta)$ -mock database generator if databases  $\mathcal{DB}_1 = \mathcal{G}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, r_1)$  and  $\mathcal{DB}_2 = \mathcal{G}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, r_2)$  are  $(t, \delta)$ -close-looking for application performance testing with overwhelming probability, where the probability is taken over all triplets  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ , and all coin tosses  $r_1$  and  $r_2$ .*

**DEFINITION 3.3.** *Let  $\mathcal{G}$  be a  $(t_0, \delta_0)$ -mock database generator,  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  be two triplets, and  $(t, \delta)$  be a pair with  $t_0(x) \leq t(x)$  and  $\delta_0(n) \leq \delta(n)$  for all  $x \in \{0, 1\}^n$ . We say that  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  are  $(t, \delta)$ -close-looking with respect to  $\mathcal{G}$  for application performance testing if mock databases  $\mathcal{G}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, r_1)$  and  $\mathcal{G}(\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle, r_2)$  are  $(t, \delta)$ -close-looking for application performance testing with overwhelming probability, where the probability is taken over all coin tosses  $r_1$  and  $r_2$ .*

Note that in Definition 3.3, it is necessary that  $t_0(x) \leq t(x)$  and  $\delta_0(n) \leq \delta(n)$ . Otherwise, we may not be able to show that a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  is  $(t, \delta)$ -close-looking to itself.

After these formal definitions, one may wonder how can we construct these generators and triplets? Indeed, the major challenging problems for generating a mock database for privacy preserving database application testing are:

1. Given function  $t(\cdot)$  and  $\delta(\cdot)$ , how to extract a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  from a live production database such that  $\mathcal{G}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, r)$  and the live production database are  $(t, \delta)$ -close-looking for application performance testing with overwhelming probability, where the probability is taken over all random coin tosses  $r$ ? The smaller  $t$  and  $\delta$ , the better. In Section 3.1, we have discussed some practical ways to extract the triplets from a live database.
2. How to exclude confidential information contained in a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ ? In the remaining part of this section, we will discuss the formal definitions of confidential information contained in a triplet and in Section 3.3, we will design practical mechanisms to exclude confidential information in a triplet.
3. Given function  $t(\cdot)$  and  $\delta(\cdot)$ , how to design a  $(t, \delta)$ -mock database generator? The smaller  $t$  and  $\delta$ , the better. In Section 3.4, we will discuss practical mechanisms to achieve this goal.

After extracting a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  from a live production database, one needs to construct a mock database for

application testing. It is straightforward that this triplet may contain some confidential information or some confidential information could be inferred from this triplet. Thus we need to exclude this confidential information from the triplet before we generate a mock database. First, we give a formal definition for the private information contained in a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ .

**DEFINITION 3.4.** *Let  $N$  be the set of positive integers and  $X = \{X_i\}_{i \in N}$  be a sequence of random variables each ranging over binary strings. A private information property (predicate)  $\tau$  on  $X$  is not contained in a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  if for any Turing machine  $\mathcal{M}$ , there is a negligible function  $\delta(\cdot)$  such that for any  $i$ ,*

$$\text{Prob}[\mathcal{M}(\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle, X_i) = \tau(X_i)] \leq \delta(i),$$

where the probability is taken over all internal random coin tosses of  $\mathcal{M}$  and over all  $X_i$ .

In Definition 3.4, we use a predicate  $\tau$  to model any information that we want to protect. A predicate  $\tau$  over a random variable  $x \in_R X_i$  can be thought as a characteristic function

$$\tau(x) = \begin{cases} 1 & \text{if } \tau(x) \text{ holds,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus  $\tau$  could be used to model any properties. The database owner may specify a list of confidential information (e.g., some security policy) that he would like to protect against database testing. It is relatively easy to convert these confidential information to a list of private information predicates (properties). Thus we can assume that the confidential information that the database owner wants to protect is a list of predicates (properties).

**EXAMPLE 1.** *Consider the following confidential information: “The average balance range of term deposits from Asian people in a specific zip code area”. We translate this property to a predicate  $\tau$ . For a 90-bit binary string  $x$ , let the first 20 bits  $x[1..20]$  represent the zip code, the next 10 bits  $x[21..30]$  represent the background of the people (e.g., 0000000001 represent Asian background, and 0000000010 represent British background), the next 30 bits  $x[31..60]$  represent the term deposit average balance lower bound for the people from  $x[21..30]$  background and  $x[1..20]$  zip code area, and the last 30 bits  $x[61..90]$  represent the term deposit average balance upper bound for the people from  $x[21..30]$  background and  $x[1..20]$  zip code area. Thus the predicate  $\tau$  could be defined by letting  $\tau(x) = 1$  if and only if the people from  $x[21..30]$  background and  $x[1..20]$  zip code area have average balance of term deposit in the interval  $[x[31..60], x[61..90]]$ .*

It is straightforward to show that for all confidential information that we have interest to protect, they could be translated into a predicate in the same way that we have done for the specific information in the previous paragraph. This shows that our definition is more general compared to the privacy definitions in statistical databases such as in [8]. Indeed, if we restrict the property  $\tau$  in Definition 3.4 to only protect exact values of certain entries in the database, then our definition is equivalent to that in Dinur and Nissim [8]. This fact could be shown by a similar proof as that in [12, 36] for the equivalence of indistinguishability and unpredictability. We will omit the details here.

**DEFINITION 3.5.** *Let  $\tau_1, \dots, \tau_m$  be a list of private information properties (predicates). We say that a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  contains no confidential information if for all  $i \leq m$ , the private information property  $\tau_i$  is not contained in  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ .*

After the above formal definition of confidential information contained in a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ , we need to design effective mechanisms to construct a new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  from the original triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  (extracted from the live production database) such that the following conditions hold:

1.  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  contains no confidential information about  $\tau_1, \dots, \tau_m$ ;
2.  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  and the live production database are  $(t', \delta')$ -close-looking for application performance testing with some acceptable functions pair  $(t', \delta')$ .

When constructing a new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  from  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ , we have to modify or delete some rules or statistical data from the original triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$ . Thus the new triplet may not be  $(t, \delta)$ -close-looking to the live production database any more (we assume that the original triplet is  $(t, \delta)$ -close-looking to the live production database). Instead the new triplet is only  $(t', \delta')$ -close-looking to the live production database. For example, a trivial approach could be to delete all rules and statistical data from  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and to let the new triplet be the empty set. In this case, absolutely no confidential information will be leaked. However, the new triplet will be  $(t', \delta')$ -close-looking to the live production database for some unacceptable function pair  $(t', \delta')$ . A practical approach would be to design an optimization mechanism so that one can gradually modify the original triplet and get the new triplet with a desired function pair  $(t', \delta')$ . It could be very hard to find a new triplet for the best function pair  $(t', \delta')$ , but some approximation to it will be sufficient in practice. In the next section, we will design effective mechanisms to construct an optimized triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  which is  $(t', \delta')$ -close-looking to the live production database and this function pair  $(t', \delta')$  is still acceptable for the application testing purpose.

### 3.3 Rule Analyzer

In this section, we discuss effective mechanisms to exclude the confidential information  $\tau_1, \dots, \tau_m$  from a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and to construct a new confidential-information-free triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ . In practice, some schema definitions, statistical data, non-deterministic rules, or deterministic rules about the real database as well as domain values for some attributes are considered as confidential information by the database owner. In particular, the confidential information property list  $\tau_1, \dots, \tau_m$  may contain the following scenarios about the disclosure of confidential information:

1. Existence of certain fields and domain values. For some tables in the live database, the existence of some fields or the name of some fields is confidential information. This is particularly true for some government databases. For example, the existence of a field for the National Security Agency in the government database had been a secret to the public for many years. It is often the case that the existence of certain domain values

in a database is confidential (e.g., Alice is a customer of Bank A or employee of FBI). Thus in the “Schema and Domain Filter”, such kind of domain values should be generated randomly.

2. Direct disclosure of some confidential rules or statistics. In some applications, some deterministic rules, non-deterministic rules, or statistics about the database are confidential information. For example, in an FBI or CIA database, some non-deterministic rules about the US spy networks in other countries are certainly high confidential information to FBI/CIA and they do not want to share this information with anyone else except among themselves. Or in a government database, the statistical data on the average year budget for army could be confidential in certain environments.
3. Indirect disclosure of confidential information. This includes: 1) some non-deterministic rules can be used to infer with high probability some deterministic rules or some statistical data; 2) some statistical data can be used to infer with high probability some deterministic rules or non-deterministic rules. If the resulting rules or statistical data are confidential, then some rules or statistics should be deleted or revised so that no information about the confidential deterministic rules would be learned from them.

For scenario 1, we have two ways to protect such kinds of privacy according to their nature: 1) Delete the corresponding field in the scheme definition and appropriately revise other related scheme definitions, rules, and statistical data if necessary. 2) Replace the corresponding field in the schema definition with a new field (e.g., a new scheme definition for a new field “comment”), and appropriately revise other related scheme definitions, rules, and statistical data if necessary. Note that it could be the case that no one except the database owner knows the existence of such a field in the corresponding table. Thus the database application software vendors are not aware of the existence of this field and the deletion of this field will not affect the test. However, it could also be the case that the database software vendors are aware of the existence of such a field, but do not know the name of the field or do not know the application purpose of this field. In this case, it should be OK to replace such a field with a new field named “comment” in the mock database.

In order to protect against direct disclosure of confidential rules or statistics as in scenario 2, the rule Analyzer will provide guidelines for database owners to decide whether each rule or statistical data should be deleted completely from the triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  or that a relaxed or completely different one should be introduced. Generally, a confidential rule or statistical data should be deleted from the triplet completely. However, in order to make the mock database more close looking, they could also be replaced with some other similar rules. In order to evaluate the different replacement mechanisms, we need to analyze its detrimental impact on the close-lookingness of the mock database and the information leakage of the new rules or the new statistical data if the new replacement mechanism is used. Optimization mechanisms should be used to find out which replacement to use.

Compared to scenarios 1 and 2, scenario 3 is relatively

hard to address. In scenario 1 and 2, one can easily find subsets  $\bar{\mathcal{R}}$ ,  $\bar{\mathcal{NR}}$ , and  $\bar{\mathcal{S}}$  of the sets  $\mathcal{R}$ ,  $\mathcal{NR}$ , and  $\mathcal{S}$  respectively, which leak confidential information in the list  $\tau_1, \dots, \tau_m$ . Then one can modify or delete entries in  $\bar{\mathcal{R}}$ ,  $\bar{\mathcal{NR}}$ , and  $\bar{\mathcal{S}}$  so that no information about  $\tau_1, \dots, \tau_m$  is leaked. However, this simple solution of deleting these identified subsets from the original triplet is not sufficient since some information in  $\tau_1, \dots, \tau_m$  could still be inferred from the remaining rules and statistical data. Special optimization methods or case specific optimization mechanisms should be designed to (automatically or semi-manually) construct a new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  from the original triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  so that no information in  $\langle \bar{\mathcal{R}}, \bar{\mathcal{NR}}, \bar{\mathcal{S}} \rangle$  is leaked in the new triplet and the mock database generated from the new triplet is  $(t', \delta')$ -close-looking to the live production database and the function pair  $(t', \delta')$  is acceptable for application testing purpose.

After the Analyzer generates the new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ , this new triplet will be fed to a  $(t, \delta)$ -mock database generator to generate the mock database. Since the mock database generator will only use random data to generate data, the view (see, e.g., [13, 12]) of any software tester (including dishonest tester) on the mock database is essentially equivalent to  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  and  $\langle \text{Schema}', \text{Domain}' \rangle$ . Since our Analyzer will guarantee that no confidential information is contained in the new triplet, our mock database is provably “secure”.

### 3.4 $(t, \delta)$ -Mock database generator

The input to  $(t, \delta)$ -mock database generator is the conflict-free and secure triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ . The output of the generator is simply the data in a flat file format which can be easily imported to databases by incorporating data schemes and domain values. We outline our process as follows.

1. Collect domain values for each attribute from the user. Those domain values such as listings of real world names of people, companies etc. are helpful to make the generated data similar to realistic. However, the database owner will provide such kind of domain values only if they contains no confidential information. Otherwise, these domain values should be generated by some pseudorandom generators. For some confidential attributes (such as names, ssn, address etc.), pseudorandom generators should be used to generate them.
2. Generate data from its statistical description for each independent attribute. The values will depend on the discovered probability distributions (e.g., uniform, gaussian, zipf, etc.). Most distributions can be derived from uniform distribution which a computer random number generator always provides (See some source code [23]). If one column has its own constraints (e.g., range constraint, key constraint, uniqueness constraint, not null constraint etc), we need to ensure the generated value satisfies the constraints. The strategy of handling single-attribute constraints was studied in [4] and can be directly applied here. Special mention should be pointed out for uniqueness constraint or key constraint over a composite attribute which consists of two or more attributes that occur together in some constraint (e.g., the pair (employeeID, departmentID) is unique in working table). We need to ensure that

each pair from the Cartesian product appears at most once among the tuples generated.

3. Apply the multi-attribute constraints to derive the values for dependent attributes. One widely used multi-attribute constraint is foreign key constraint (which may across different tables). When generating a value for attribute  $A$  in relation  $R$ , where this attribute references attribute  $A'$  in relation  $R'$ , we need to insure the value for  $R'$  is generated before that of  $R$ . The other multi-attribute constraints are usually derived from semantic or business rules. Some of them may be explicitly included in the schema (e.g., functional dependencies, check, assertion, trigger etc.) while the other may come from applications. Special mechanisms may be designed to automatically derive the values for dependent attributes by following an optimal order of constraints.
4. Import data files to mock database by incorporating filtered data schemes and domain values. As we discussed in Section 3.2, data schemes and domain values may contain confidential information which needs to be filtered before incorporating into data generator.

It is imperative that we generate data effectively which follow all given rules and statistics. However, the above process has one potential drawback that the generated data may violate some non-deterministic rules which will make the generated data not close looking to real data. In general, to generate data which satisfy both constraints and statistics is a combinatorial optimization problem. Heuristic techniques such as iterative relaxation method, constraint solving techniques [15], simulated annealing method [20] etc. could be used to achieve this task.

In this paper we apply simulated annealing approach to arrive an approximate optimal result. We include this before importing data to mock databases. The idea is to swap attributes between tuples to satisfy all possible multi-attribute rules (including non-deterministic rules). The general principles we follow here include: applying the minimum changes, satisfying as many as possible rules, and modifying as less as possible the original frequency distribution of the data. For each rule  $r_j$ , we assign  $CF(r_j)$ , a cost function which measures how far the data is from satisfying the rule imposed on the data by a user. Our goal here is to minimize  $\sum_{r_j \in R \cup NR} CF(r_j)$ . This method converges to the minimum of cost function in a time polynomial in the size of the relation. It is no surprise that this new step involves scalability issues for large data sets.

It is imperative that we generate data efficiently. Gray and Sundaresan [16] investigated techniques how to quickly generate a large database by using parallel algorithms and execution given the statistical distribution of the underlying data. Their approach can also be applied here. Our case is more complex as the generated data need to satisfy both the given rules as well as statistical distributions. Our interest here is to efficiently generate reasonably close looking data for all of the tables in database, representing all of the important characteristics that have been identified.

### 3.5 System implementation

In response to a lack of existing tools specifically designed for testing database applications, we are designing and implementing a prototype system to automate the process of

creating and populating the mock database based on the rules and statistics.

The input to our prototype system includes database schema definition, along with additional information (e.g., rules, statistics) from the user. The schema definition for the database underlying the application to be tested is parsed by an SQL parser while the rules are analyzed and parsed by a rule Analyzer. There are several open-source SQL parsers available (e.g., PostgreSQL [27]) and we will integrate one into our prototype system to extract information from schemes which are defined by DDL. The SQL parser will create a parse tree that contains all the relevant information about the tables, attributes, and constraints, etc. The core part of our prototype system is Rule Analyzer which contains two separate components: conflict resolution and disclosure assessment.

This prototype system will also integrate many data perturbation methods and randomized functions (e.g., uniform distribution, multi-dimensional Gaussian distribution) which are used to generate synthetic data and data mining methods which help to extract rules and statistics from data. In our previous discussion, we assume that the  $(t, \delta)$ -mock database generator does not need to access the individual data. However, if the generator can access the live data, we may investigate the *Perturbation* approach where the sensitive values in a user's record will be perturbed using a perturbation function so that they can not be estimated with sufficient precision. For example, we can apply additive noise approach [18, 19] for numerical attributes<sup>2</sup>. The additive noise approach is a very effective approach for masking multivariate normal data (the underlying real life data are often assumed to be approximately multivariate normal) that preserves confidentiality and can preserve some essential characteristics of the data such as means, variances, and correlations. The noise can be done using gaussian or uniform distributions. One of the major advantages of the approach is that it allows one to obtain precise subpopulation estimates while keeping the correlation structures of live data.

This prototype system will also provide a controlled environment in which diverse data can be experimentally generated and diverse techniques can be evaluated. We aim to investigate the interactions between the values used to populate the database and the values the tester enters as inputs to the application program. As the output of database application includes the entire database instance which will generally be large and complex, we need to develop components of efficiently validating the output of test cases as they are executed.

In order to evaluate our approach, we will do experiments in the following directions and compare the results.

- Database application testing running on the original database;
- Database application testing running on  $(t, \delta)$ -close-looking mock databases for various values of  $t$  and  $\delta$ ;
- Database application testing running on randomly generated databases; and

---

<sup>2</sup>For those sensitive non-numerical attributes such as SSN, name, address etc., the lookup tables together with some cryptography techniques can be applied here.

- Database application testing running on duplicated live database with privacy preserving filters.

#### 4. CONCLUSION AND ADDITIONAL RESEARCH ISSUES

We presented a solution for privacy preserving database application testing. Our solution consists of the following steps.

1. Extract a triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  from the live production database such that a mock database generated from this triplet is close-looking to the live production database for database application testing purpose;
2. Exclude confidential information from the triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  and construct a new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  such that this new triplet contains no confidential information about the live production database and a mock database generated from this new triplet is also close-looking (in a relaxed form) to the live production database for database application testing purpose;
3. Use a mock database generator to generate a mock database from the new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$ .

Formal definitions to database close-lookingness and private information leakage have been given and the relationship among them have been studied.

There are some aspects of this work that merit further research. The live production database is evolving all the time and the privacy policies and rules may be changed over the time. Thus the triplet  $\langle \mathcal{R}, \mathcal{NR}, \mathcal{S} \rangle$  for the live database is changing all the time. In order to keep the mock database update, we will study incremental building mechanisms so that we do not need to rebuild the mock database completely. At any time that we need to update the mock database, we collect the updated triplet for the live database and construct an update new triplet  $\langle \mathcal{R}', \mathcal{NR}', \mathcal{S}' \rangle$  for the mock database. Another aspect that merits further research is to build distributed mock databases for application software testing as distributed databases occur in many scenarios. How to extend approaches to such environment needs further study.

#### 5. ACKNOWLEDGMENTS

The work was supported in part by U.S. National Science Foundation NSF CCR-0310974. We would like to thank the referees for the detailed comments on improving the presentation of this paper.

#### 6. REFERENCES

- [1] N.R. Adam, and J. C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21(4):515-556, Dec. 1989.
- [2] R. Agrawal, and R. Srikant. Privacy-preserving data mining. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pp. 439-450, Dallas, Texas, May 2000.
- [3] L. Brankovic, and V. Estivill-Castro. Privacy issues in knowledge discovery and data mining. In *Proceedings of 1st Australian Institute of Computer Ethics Conference*, July, 1999.

- [4] D. Chays, S. Dan, P. Frankl, F. Vokolos, E. Weyuker. A framework for testing database applications. In *Proceedings of International Symposium on Software Testing and Analysis*, Portland, Oregon, August 2000.
- [5] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *FOCS* 1995.
- [6] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *JCSS* 60(3): 592-629 (2000).
- [7] R.A. Davies, R.J. Beynon, and B. F. Jones. Automating the testing of databases. In *Proceedings of the first International Workshop on Automated Program Analysis, Testing and Verification*, June 2000.
- [8] I. Dinur and K. Nissim. Revealing information while preserving privacy. In: *Proc. 22nd ACM PODS*, pages 202–210, ACM Press, 2003.
- [9] J. Domingo-Ferrer. Current directions in statistical data protection. In *Proceeding of Statistical Data Protection*, 1998.
- [10] V. Estivill-Castro, and L. Brankovic. Data swapping: balancing privacy against precision in mining logical rules. In *Proceedings of International Conference of Data Warehousing and Knowledge Discovery*, 1999.
- [11] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy Preserving Mining of Association Rules. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.
- [12] O. Goldreich. *Foundation of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [13] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Computing* 18:186–208, 1989.
- [14] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science* 28(2):270–299, 1984.
- [15] A. Gotlieb, B. Botella, and M. Rueher. Automatic test data generation using constraint solving techniques. In *Proceedings of the 1998 International Symposium on Software Testing and Analysis*, pp. 53-62, March 1998.
- [16] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-records synthetic databases. In *ACM SIGMOD*, pp. 243-252, June 1994.
- [17] M. Kantarcioglu, and C. Clifton. Privacy preserving distributed mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 24-31, June 2002.
- [18] J.J. Kim. A method for limiting disclosure in microdata based on random noise and transformation. In *Proceedings of the section on survey research methods*, American Statistical Association, 1986.
- [19] J.J. Kim, and W.E. Winkler. Masking microdata files. *Report of Bureau of the Census*, 1997.
- [20] S. Kirkpatrick, S.D. Gelatt, and M.P. Vecchi.

- Optimization by simulated annealing. *Science* 220(4958):671-680.
- [21] Y. Lindell, and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, pp. 36-54, 2000.
- [22] Niagara. <http://www.cs.wisc.edu/niagara/datagendownload.html>
- [23] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C, the art of scientific computing*. Cambridge University Press, 1988.
- [24] Quest. <http://www.almaden.ibm.com/software/quest/>
- [25] S. Rizvi, and J. Haritsa. Privacy preserving association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*. Aug, 2002.
- [26] C.J. Skinner. On identification disclosure and prediction disclosure for microdata. *Statistica Neerlandica*, 44:21-32, 1992.
- [27] M. Stonebraker, and L. Rowe. The design of postgres. In *Proceedings of ACM-SIGMOD International Conference on the Management of Data*, June 1986.
- [28] B. Malin, L. Sweeney, and E. Newton. Trail re-identification: learning who you are from where you have been. *Proc. LIDAP-WP12*. Carnegie Mellon University, 2003.
- [29] Transaction Processing Performance Council. TPC-Benchmark C. 1998.
- [30] Edward Tsang. Foundations of constraint satisfaction. Academic Press, 1993.
- [31] J. Vaidya, and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.
- [32] J. Vaidya, and C. Clifton. Privacy preserving k-means clustering over vertically partitioned data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 206-215, August 2003.
- [33] G. Wiederhold, and M. Bilello. Protecting inappropriate release of data from realistic databases. In *Proceedings of the Ninth International Workshop on database and Expert Systems Applications*, Vienna, Austria, 1998.
- [34] G. Wiederhold, M. Bilello, and C. Donahue. Web implementation of a security mediator for medical databases. In *Proceedings of the Eleventh International Conference on Database Security*, 1997.
- [35] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE symposium on Foundations of Computer Science*, pp. 162-167, 1986.
- [36] A. Yao. Theory and application of trap-door functions. In *Proc. of 23rd IEEE Symposium on Foundation of Computer Science*, page 80–91, 1982.