## Testing Health-care Integrated Systems with anonymized test-data extracted from Production Systems

Ali Raza

Computer Science Department

Utah State University

Logan, Utah

ali.raza@aggiemail.usu.edu

Stephen Clyde

Computer Science Department

Utah State University

Logan, Utah

Stephen.Clyde@ usu.edu

***Abstract—*Testing of data-centric health-care integrated systems involve numerous non-traditional testing challenges, particularly in the areas of input validation, regression testing, and load testing. For these and other types of testing, the test-data suites typically need to be relatively large and demonstrate characteristics that are similar to real-data. Generating test-data for integrated system is problematic because the records from the different systems need to be inter-related in realistic and less-than perfect ways. Using real-data is also not a feasible choice, because health-care data contains sensitive personal identifying information (*PII*). This paper presents our experiences with a test-data creation tool that extracts loosely correlated slices of data from multiple operational health-care systems and anonymizes that data such that it contains no deducible *PII*, but still preserves those real-data characteristics that are important for effective testing. As a foundation, this paper also provides a classification of testing challenges for health-care integrated systems and a comparison of anonymization techniques.**

Keywords-Data anonymization in Health-care, test-data extraction, integrated-systems, testing e-Health Systems, test-data creation, data-quality issues in e-Health care.

# Introduction

Child Health Advanced Record Management (*CHARM*) is an integrated system that provides health-care professionals with accurate and timely information about children in Utah, whose medical records are housed in various federated public healthcare databases, including Vital Records (VR), the Utah State-wide Immunizations Information System (USIIS), and Early Hearing Detection and Intervention (HiTrack). The first functional prototype was successfully demonstrated in March 2002. It was at this point that the developers began to see the real challenges of testing an integrated system that involves confidential data. With the three original participating programs, the system made use of seven different databases: three from the participating programs, three used by the agents to map PP-specific IDs to internal *CHARM* IDs, and one used by the *CHARM* server to match and link persons based on their demographic information[7]. From a testing stand-point, however, such data separation made generating realistic test-data difficult.

At first, the developers tried to create test-data by hand. This quickly proved to be time consuming and error prone, resulting in test-data that was neither realistic nor effective in detecting software failures. Next, the developers built an automated test-data generator (*TDG*)[1, 12] that created test-data for each database using that database’s scheme and codified knowledge about field domains, constraints, and overall data characteristics. The *TDG* approach allowed developers to create large amounts of test-data, but correlating the records between different databases and creating patterns similar to those in the real-data proved difficult. We learned the *TDG*s work well for smaller and simple data-schemes, but are awkward to configure large complex integrated systems and resulted test-data is not as effective we would have liked. For example, consider the simple case of generating meaningful test addresses. Below are some sample generation rules for realist address data:

* The state must be a real state
* 95% of the cities must be a real city in the state
* 90% of the street addresses must be a real street address
* If the city and street address is real, then zip code must be real for that combination about 90% of time.
* 2-3% of cities should be blank
* 4-5% of the street addresses should be blank

To create better quality test-data, we need to add even more rules. However, as we add more rules, the generation logic becomes more complex and the *TDG* execution time increases.

Now, on top of this, add the complexity of ensuring that a certain percentage of test records from multiple data sources that are supposed to be correlated have reasonable similar addresses. In our research and literature review [1,2,3,4,5,6,7,8,9,10,12], we did find any *TDG* that can claim to generate realistic test-data for integrated environments.

After trying *TDG*’s, we turned our attention to Test-data extraction (*TDE*), which creates test-data from existing, operational data sources. We anticipate that *TDE* would prove a better alternative, because it provides more realistic test-data for confronting challenges related to testing health-care integrated systems (section V). Additionally, configuring *TDE* is much easier and faster than *TDG*. However, *TDE* also has some limitations such as data-privacy concerns, resolving data heterogeneities, long running-time for data extraction, interruptions due to network or database problems and varying access privileges. Among them, preserving data-privacy [21, 22, 23] especially *PII*’s (section II) and data heterogeneities [20] are considered the hardest problems. There exist different tools and techniques (section III) which can help to anonymize *PII*’s in real-data. In this paper, we also provide a classification scheme (section IV) of important health-care data characteristics in integrated testing. A matrix that compares ‘capability to preserve data-semantics’ of different anonymization techniques against this classification is given in section IV. We described a *TDE* tool, called *iSTDE*, which we used to test *CHARM* (section V). Section VI talks about experiences with *iSTDE*. Summary and references are provided in section V11 and V111 respectively.

# Data Privacy and *PII*

Data Privacy, in context of health-care, defines challenges related to sensitivity of personal identifying information to external data-threats. Different regulations such as *Health Insurance Portability and Accountability Act (HIPAA)* [15] and *National Institute of Standards and Technology (NIST)* [16] attempted to characterize chunks of information which can define the identity of a person and termed it as *Personal Identifying Information* or *PII*s. Recently protecting the personal privacy got a huge momentum under today's information world. It has become one of the primary focuses of *HIPAA* and *NIST*. They also provide a list of *PII*’s. In general, there are two ways of hiding the *PII*s. Data de-identification and data anonymization.

*Data de-identification* relates to the information from which *PII*s have been removed or obscured and the remaining information can't identify any person. However the de-identified information can be re-identified by using a code or a program. *Data anonymization* is also about obscuring *PII*’s, but in this case, there exist no code from which information can be re-identified. In health-care, most regulatory authorities prefer data anonymization to data de-identification, because de-identification would always pose a serious threat to privacy and misuse of information.

# Tools and techniques for Data Anonymization

Data Anonymization tools and techniques fall under following broad categories; data-generalization, data-suppression [23], adding noise and data swapping [25]. In data generalization, we replace the identifying data with more general, less specific value, while keeping a correlation to the original. For example zip code value 84321 can be replaced with 8432\* (level 1), 843\*\* (level 2) and so on. Similar to numbers, categorical attributes can also be generalized, provided a domain generalization hierarchy. For example, ‘Albert’ a person name can be generalized to ‘American’ (Level 1) or ‘Male’ (Level 2), supposing generalization hierarchy is of the form (Albert 🡪America 🡪 Male 🡪 Person). In data suppression we do not release the values at all for suppressed columns. Introducing noise is related to adding small amount of variations into selected data. In data-swapping, we swap a field of one record with a field from another record. The approach we experienced and discussed in this paper is a specialization of data-swapping termed as *‘semantic data-swapping’*, where we swap the *PII*’s, while keeping the data-dependencies intact, which don’t disturb overall semantics of records.

Various research-based techniques for data-anonymization include but not limited to datafly, u-Argus, k-anonymity (AllMin, MinGen, Bottom-up Generalization, Top-down specialization, K-Optimize, Incognito, Mondrian[21, 22, 23]) and *iSTDE*[13]. All these techniques use one or more broad categories of data generalization, suppression, noise-addition or data-swapping.

# Classification of test-data characteristics in Health-care Integrated Systems

This section defines real-data characteristics which are important for testing integrated systems in health-care. Comparisons with respect to semantic heterogeneities, schematic heterogeneities, field-level data quality problems, record-level data-quality problems and overall capability to preserve data-privacy are provided in section V. Table 1 summarizes the classification of real-data characteristics and comparison of anonymization techniques.

## *Semantic Heterogeneities*

### *Similar fields with different semantics:* Sometimes values in two fields appear similar but their interpretations may be different. For example, mailing address and current address respectively. Testing of matching problem in integrated systems may need a frequency distribution of similar or different current/mailing addresses to identify persons.

Semantic-changes due to domain evolution: Database schemes (syntactic) and data definitions (semantics) evolve over time as organizations grow or shrink. For example, in a hospital inventory, we can expect an alternate code scheme for surgery equipment. Initially, say ‘H01’ code was used to represent equipment ‘A’; later it was assigned code ‘H011’. The evolved code scheme value may be stored in the same field or in a new field. Intersting test case for Merger would be to check how accurately it detects and interpret these semantic-changes due to domain evolution.

Incompatible code names: Data sources in integrated environment may have different coding schemes to represent the same information. For example, persons in two data sources can use slightly different, or incompatible ethnicity codes to represent person’s ethnicity. Testing of Merger/Resolver in integrated systems, needs such incompatible codes along with their sources so that they can test various strategies to resolve the conflicts.

Impossible/Meaningless values: Often times, database designers or users assign impossible/meaningless values to represent a particular type of information. For example, users can assign -1(impossible value) to represent weight for a baby in a particular category, or a meaningless value such as "boy" or "girl" to represent baby temporary name. These sorts of values may be the result of week user interfaces, design of database-schemes, user preferences, or some errors. Similar to incompatible codes, Merger/Resolver often have pre-defined rules to deal with these impossible/meaningless values. Testing those rules is only possible with data having certain percentage of impossible and meaningless values.

Extra-information: In this heterogeneity, extra information is attached with the record to convey some meaningful information or message. One such example is ‘Henry John (deceased)’ which is conveying the notion that this person has died. Merger is supposed to extract the common form of information from the record and handle it. For example, it should link ‘Henry John’ with the name and ‘deceased’ with the death information. In CHARM, testing of SyncEngine needs such extra information.

## *Data Problems caused by Schematic Heterogeneities*

Schematic heterogeneities are about differences in database schemes or the way information is represented. Few important heterogeneities are described below.

### *Representation with single-table vs. multiple-tables:* One data source is representing addresses inside *PERSON* table whereas other data source is using a separate *ADDRESS* table (link to *PERSON* table) to represent same address information.

Representation with one-column vs. many-columns: One data source is using one-column whereas another data source is using three-fields to represent date of birth information.

Representation using one-record vs. many-records: Daily transactions or logs can be maintained either horizontally or vertically in a database table. In vertical-fashion, one record can represent one full piece of information, whereas in horizontal-fashion, one record can represent just one of the many pieces of information. Row-wise dependency/column-wise dependencies further elaborate these thoughts. A row filters possible sets of values in another row. Row-wise designed tables add the flexibility in database schemes to escape refactoring efforts for possible schematic changes in the table’s attributes. For

example, we can see this dependency in the HISTORY\_CHANGES table of the VS database that maintains logs related to changes in its PERSON table. Here, for one person, there exist multiple rows in the table that describe some information about that person, i.e., one row for gender and one row for first name, etc.

In integrated systems, data-sources often contain similar information with different representations (usually one of the three types discussed above). These real-data characteristics explore schematic testing of heterogeneous data sources in health-care systems. During matching, linking and merging processes, components should have information about how to map, match and resolve the conflicts (with various representations).

## *Field-level Data Quality Testing Problems*

### *Degree of Variations*

### We identified four different kinds of name variations that can provide a rich set of test cases to perform input validation testing.

*a. Spelling variations.* Sometimes a name can be spelled in many different ways. For example, ‘jimmy’ as a first name in *PERSON* table can be spelled as ‘jimmy’, ‘jiimy’, ‘Jimmy’, ‘JIMMY’, ‘JIMI’, or ‘jimi’, etc. Checking spelling variations is an important test case for examining the accuracy of matching algorithms.

*b. Complete and incomplete names.* We also need a combination of complete and incomplete names to test *Matcher*. An incomplete name like ‘Jo’ can be a substring for many complete names like ‘John’, ‘Johnson’, ‘Joddy’, etc. Another example is where many persons have the same first name, for example, ‘Byron’, but have different last names, such as ‘Byron Douglas’, ‘Byron John’ or ‘Byron Robert’.

*c. Case variations.* In actual data, we also find some case variations in first name and last name combinations like ‘Abdullah Hassan’, ‘ABDULLAL HASSAN’, ‘Abdullah HASSAN’. Identification of similar person records among name variations in different databases can be a very powerful set of test cases for the *Matcher* component.

*d. Dummy names and garbage data.* Not only should the testbed contain variations in actual names but also different types of dummy values. For example, ‘DJ’, ‘Jim321’,’Girl’, ‘Boy’, etc., names with special characters like ‘To'e’, ‘Wall-J’, ‘Olivas-Parez’ and ‘Pulefa’alii’ or garbage data such as ‘A’, ‘aaaa’, ‘@’,’^’, etc. A common testing technique to find validation bugs is by processing a rich set of dummy names and garbage data.

Data Dependencies

Functional Column Dependencies can be of three types.

*a. One-way data dependency.* In table T, column A determines the value for another column B or *A🡪B*. One-way data dependency exists between *first name* and *gender* columns in a *PERSON* table, where *first name* determines *gender*. In *CHARM*, the *Matcher* component uses a father name to search for his children. In the absence of one-way data dependency which distinguishes a father from a mother, our component could not find any children, and thus would be unable to verify matching-related test cases.

*b. Bi-directional Data Dependency.* For table T, columns A and B are dependent on each other, or *A🡪B* and *B🡪A.* We can see bi-directional data dependency between state code and state name columns of the *US States* table, where both columns determine values for each other.

*c. Aggregate Data Dependency.* A dependency exists among a group of columns, say *C1, C2, C3 … Cn*, for Table T such that individually they have bits of bigger chunk of information that needs to treated as a whole. Consider an *ADDRESS* table that contains five columns: *street\_address1, street\_address2, state1, zip1, city1*. These fields have an aggregate data dependency among themselves, i.e., the value for one of individual fields is partially dependent on one or more values from other fields. Testing of the *CHARM*’s *AddressCleaner*, for example, requires test-data that models this aggregate data dependency. The *AddressCleaner* takes the unclean address (partial, incomplete, or wrong address) as input and returns a cleaned address (a complete address). The addresses are searched and matched against a huge address database, maintained by an external system through an edit-distance approach, i.e., the more complete the address is, the more likely it exists in the database. *AddressCleaner* cannot be tested if test-data do not have aggregate data dependency among the address fields.

referenced values

We identified two kinds of field-level referenced values which are highlighted below.

a. Explicit referential-integrity constraint. Also known as primary-foreign key relationship, this is considered an important data characteristic for testing database applications, as many times we need to test the applications for cascading inserts, deletes, and updates to ensure that data values are synchronized among tables.

b. Indirect referential-integrity constraint.This is an implied value-based inter-table reference without a {primary/foreign key} relationship. This constraint is a kind of hidden data-dependency. Hidden dependencies can be a result of missing an explicit referential integrity constraint or incomplete database refactoring when data in tables already exist. For example, in CHARM CORE database, both PHONE NUMBER and ADDRESS tables have an implied reference with PERSON table because a phone-number value of a person hints his address. A phone number value ‘435-797-3786’ can suggest that the owner has a Utah-based address entry in the ADDRESS table. AddressCleaner component of CHARM has some inbuilt rules, similar to this characteristic. Testing of those rules need data with indirect referential-integrity constraints.

Support for Derived Data

*CHARM* dataset defines two types of derived data: computed field values and restricted values to a computed set.

a.Computed field values. This field derives its value from two or more other fields or A=f (B1…Bn). For example, an employee’s salary is computed from a set of fields, such as gross salary, tax, deductibles, leave taken, etc. Although computations can be done at application-level, sometimes it becomes mandatory to perform such computations at database-level behind some procedure or trigger. A reason for having computed fields in the database schemes is to reduce the query response time. Much important query processing time can be saved by providing already computed values as separate database columns that can be populated or computed from some trigger in a regular pattern.

b.Restricted value to computed set. This category consists of values that have partial independent values and partial computed values. In a mathematical expression, this characterization can be represented as A=Random Selection (f (B1…Bn)). For example, a contact name field can contain a brother name or some relative name, which is some sort of restricted value with respect to person name. It might be possible that two brothers or relatives have the same last name.

Contains Data Frequency Patterns

In a database, we can expect two different kinds of data-frequency patterns.

a.Unique Values. Unique values are normally associated with rows for identification purposes. There exist a number of ways to generate unique values, for example, through sequence numbers, unique random numbers, semantics-related unique values, global unique identifiers (GUID), etc., each having its own varying time and complexity. Both standalone database testing as well as integration testing depends heavily on unique values.

b.Frequency distribution of data values in single/multiple column(s). For single column, this characteristic determines the frequency distribution of data values or a pattern of different values in a single column. For example, FIRST NAME column in PERSON table of CORE database can have 30% female names, 50% male names, and remaining 20% as null values. In multi-columns two or more columns share a common frequency-based pattern of data values. For example, PATIENT NAME and IMUNIZATION IDENTIFIER columns in FORECAST table of the USIIS database have a multi-column data value frequency distribution. These columns have different values, but the percentage of distribution is the same. In CHARM, data frequency patterns can raise bugs related to matching. For example, to find number of twins, Matcher identifies multiple birth flags for possible twin-matches. A frequency-based distribution pattern for twins that is close to reality can identify potential bugs in Matcher.

## *Record-level Data Quality Testing Problems*

### *Duplicate records:* A data-source may contain duplicate records, though the two records may have different identity-keys.

Incorrect records: A record is said to be incorrect if one or more of its fields do not observe certain kind of data-dependency. For example, a person record has a city value ‘SAN DIEGO’ but state ‘NEW YORK’.

Incomplete records. A record is incomplete if PII’s for one or more fields are missing. Example is a person who has street-address but city,state and zipcode are missing.

Matching algorithms in integrated systems work on various approaches to find an exact match, or probabilistic match for a person. Testing these kinds of matching rules in integrated systems need data sources that contain sufficient instances of duplicate, incorrect or incomplete records.

## *Capability to preserve Data Privacy*

We found that data-swapping would retain more real-data characteristics such as (percentage of missing, incomplete, incorrect data-instances both at record-level as well as field-level) than other anonymization techniques such as generalization, suppression and noise addition. However we also identified that suppression and generalization are better techniques for preserving data privacy than swapping (see challenges in section VII).

# Comparison of different Data Anonymization Techniques under Classification Scheme

We analyzed four data-anonymization techniques under two different contexts; preserving data-privacy and data-semantics respectively. The more we include data-semantics in our test-data, the less secure it would be. For example, real-data preserves maximum data-semantics but it is also very susceptible to data-privacy threats. In general, we can place four data-anonymization techniques in the following order with respect to the capability to preserve data privacy (from lowest to highest): real-data, semantically-swapped data, generalized-data, data-with-noise and suppressed-data respectively.

In our analysis, ‘semantic-data swapping’ fully preserved all data characteristics of classification because data-shuffling is performed without affecting data dependencies. Noisy-anonymization partially preserved where as data generalization and suppression didn’t preserve majority of semantic-dependent characteristics. In integrated data-environment, it is very difficult to create a generalization hierarchy because datasets are huge, have incompatible values with varying semantics of similar concepts. Additionally, by suppressing important *PII*’s, we can preserve data-privacy but also loose frequency distribution of data-semantics. ‘Data anonymization by noise-addition’ can preserve data-semantics to some degree but also increases threats to data privacy.

At the highest level of classification scheme, schematic data-problems are found to be least affected by data-anonymization techniques because schematic heterogeneity is all about data-representations rather than something to do with data-semantics.

Out of total eight comparisons (in classification scheme) on data-anonymization techniques, semantic data-swapping preserved all eight data characteristics. Noise-addition preserved four full and four partial. Two fully preserved and three partially preserved data characteristics are supported by data generalization. In the end, data suppression preserved full semantics for just two data characteristics and partial for one. While data suppression proved to be the least-semantic preservation technique, it was considered the safest in terms of preserving data-privacy with little or no-risk. Analysis on individual comparison on each data characteristics for testing health-care system is provided in table 1.

1. A Comparison of Anonymized Data Approaches in Integrated Data-centric Environment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristics of Real-data that need to be Preserved**  **For Testing Integrated Health-Care Systems** | **Data Anonymization Techniques** | | | |
| **Semantic-Data Swapping** | **Adding Noise** | **Data Generalization** | **Data Suppression** |
| A. Data Problems caused by Semantic Heterogeneities | | | | |
| 1. Semantic difference in similar fields  2. Semantic-changes due to domain evolution  3. Incompatible code names  4. Impossible/Meaningless values  5. Extra-information |  |  |  |  |
| B. Data Problems caused by Schematic Heterogeneities | | | | |
| 1. Representation with single-table vs. multiple-tables  2. Representation with one-column vs. many-columns  3. Representation using one-record vs. many-records |  |  |  |  |
| C. Field-level Quality Problems | | | | |
| 1. Degree of Variations  a. Spelling variations  b. Complete and incomplete names  c. case variations  d. Dummy names and garbage data |  |  |  |  |
| 2.Data Dependencies  a.One-way data dependency  b.Bi-directional data dependency  c.Aggregate data dependency |  |  |  |  |
| 3.Referenced Values  a.Explicit referential integrity constraint  b.Indirect referential integrity constraint |  |  |  |  |
| 4.Support for Derived Data  a.Computed field values  b.Restricted value to computed set |  |  |  |  |
| 5.Contains Data Frequency Patterns  a.Unique Values  b.Frequency distribution of data values in single/multiple column(s). |  |  |  |  |
| D. Record-level Data Quality Testing Problems | | | | |
| 1. Duplicate records  2. Incorrect records  3. Incomplete records |  |  |  |  |
| E. Preserve Personal Identifying Information *(PII’s)* | | | | |
| Overall capability to preserve Data Privacy | Moderate Risk | Moderate Risk | Minor Risk | Little/  No Risk |

|  |  |
| --- | --- |
|  | Fully preserved semantics |
|  | Partially preserved semantics |
|  | Semantics not preserved |

# *iSTDE*, Semantic-based extraction of anonymized test datasets in Integrated Environment

In the latest version of *CHARM*, the developers have taken a new approach for creating test-data. Specifically, they created a distributed tool, called Semantic Test-data Extractor for Integrated Systems (*iSTDE*) [13], which extracts an anonymized and consistent cross-section of data from the production databases. It next manipulates the data in a way that obscures individual identities, while preserving other important aggregate data characteristics, such as the frequency of name occurrences, the percentage of multiple births (i.e., twins), and the presence of bad data. Preserving these characteristic is critical to effective system testing of components like a *Person Matcher*. After de-identifying the test-data, *iSTDE* moves that test-data from the production environment to a test environment. More details about the architecture and methodology of *iSTDE* are available at [13].

*iSTDE* software is installed in a confidential environment, thereby ensuring that no unauthorized person can execute it. It goes through a six-step process to create a consistent set of anonymized test-data.

In the first step, a user specifies what data to extract (e.g., all children born from 7/1/2008 to 9/30/2008) and the target environment description wherein test-data should ultimately be sent.

The second step in the *iSTDE* execution involves creating the temporary database in a confidential environment to hold the extracted data from multiple source databases, while they are being collected and manipulated.

Now in this the third step, *iSTDE* extracts a consistent slice of real-data from the participating data sources in the confidential environment and loads that data into temporary databases, created in second step.

The fourth step is data mangling. In the first phase, *PII*’s domains and their internal dependencies are identified. After *PII*’s domains selection, we build dictionaries (with domain dependencies) for these domains. These domain dictionaries are data structures that consist of real domains and test domains. Test domains are built using random or semi-random shuffling of real domains. In the next mangling phase, we swap all the values of *PII*’s domains in real-data with the newly assigned test values using domain dictionaries that provides mapping from real values to test values. In other words, the original values with dependencies are randomly shuffled with a new value having similar sort of dependencies.

Once mangling of data is complete, the fifth step in the entire process is the automatic transfer of anonymized test-data to user-specified unprotected environment.

Final step destroys all traces i.e., data files and domain dictionaries that could identify or even hint at any sensitive information about the patients.

# Experience with *iSTDE*

In the last three years, we have used *iSTDE* to test several key *CHARM* components including *Matcher*, *AddressCleaner*, *SyncEngine* and *Web Interface*. It was particularly useful in testing the *SyncEngine* because its primary function is to correlate data among multiple data sources in *CHARM* (Participating Programs). Previously, testing the *SyncEngine* required developers to create test-data manually. Not only was this very time consuming, but it resulted in test-data sets with limited coverage. *CHARM* developers found datasets produced by *iSTDE* to be good approximation of the real-data with properly hidden personal identities. Most importantly, the *CHARM* developers found several critical software faults using the *iSTDE* test-data that would have been difficult to find with handcrafted test-data or with any *TDG* approach. The *CHARM* developer had similar experiences testing the *CHARM* *Matcher* and *Web Interface*. In the following paragraph, we would discuss few recommendations, challenges and hard issues we experienced in our test-data anonymization process using *iSTDE* in *CHARM*.

## *Recommendations*

Our experience with the iSTDE suggested that data-privacy concerns, and time to extract test-data are two major problems. We observed few practices which proved very helpful in this regard.

### Threats to privacy in *iSTDE* can be enhanced by a) running test-data anonymization process in confidential environment and then transfer mangled data in unsecure environment. b) implementing views as tables if anonymized data is not supposed to include all data source tables.

TDE data-extraction time is not dependent on characterization rules, but it is dependent on the volume of data, federated data sources, load on database managers, query-joins, and network-speed. In CHARM, iSTDE takes almost two days to extract a two-week correlated slice of test-data from seven data sources. Time to extract can be reduced significantly by a) concurrent execution of test-data extraction processes, query optimizations and scheduling processes in off-hours. b) by using anonymization as process rather than a tool, and manging repository of anonymized testbeds which can later be re-used to avoid costly test-data extraction time.

## *Challenges*

We also came across few interesting challenges and solutions while extracting anonymized test-data slices from *iSTDE*.

### If there exist just one record in a table, mangling is not possible and *PII’s* would not be obfuscated. We made some changes in new version. Domain dictionaries are not deleted from now onward. These dictionaries are evolved continuously overtime as record-set in production data-sources increases. However they are shuffled periodically, so that data-shuffling approach remains anonymized rather than de-identified. Hence it reduces occurrences of mangling for just one record.

You should expect different levels of access permissions in different data sources. For example, you may get a ‘read-only’ permission to just access the data and not its meta-data. We overcome this limitation by generating meta-data from SQL queries and maintaining meta-data information in a repository for reuse.

In data-integration environment, you may also notice differences in data-types due to heterogeneities in database managers. We transformed these type-based heterogeneities by maintaining a dictionary of conversion rules.

## *Hard Issue/Limitations*

So far in our experience with *iSTDE,* we are still struggling with the following hard issues.

### Obfuscating *PII*’s would not always ensure data privacy preservation. Every non-*PII*’s attributes describe something about record. Using these characteristics along with some prior knowledge, researchers have successfully re-identified *Netflix* users’ movie ratings [24].

Heterogeneities in non-data containers such as database procedures and triggers can’t be resolved. Different database managers use different procedural languages and it is very difficult to automate this implementation. At maximum, we can either implement dummy procedures or can store them in a repository after manual translation to desired procedural language.

Data sources in integrated systems evolve independently of each other. Changes do not occur in a lock-step chronology. One system can upgrade its database without publishing its changes to others. Under these circumstances, if one data source implement some schematic changes and do not provide access to metadata, test-data extraction for those database objects would not succeed.

In extracting consistent slice of data from different data source managers, it is not uncommon to expect certain abnormal behaviours such as network-down, temporarily database shut-downs, slow query processing time and many others. Such kinds of interruptions can cost huge time or produce inconsistent test-data slices, while extracting large test-data sets.

iSTDE extracts and anonymized a synchronized test-data set from heterogeneous data sources to a homogeneous environment. Hence we can’t test applications for syntactic heterogeneity such as differences in database managers, query languages and other database objects respectively.

# Summary

This paper narrates our experiences about extracting and anonymizing test-data using *iSTDE* from various production data sources in *CHARM*. We explored various techniques of creating test-data and discussed their advantages and limitations. We also provide a classification scheme of real-data characteristics which are important for testing health-care integrated systems and then provide a comparison of different anonymization techniques against this classification. Later we identified few guidelines, interesting challenges and some hard issues while working with *iSTDE* over a period of three years. Overall we found that performance of semantic-data swapping using *iSTDE* is preferable to other extraction and anonymization techniques with certain limitations, which, to some extent, can be overcome using a well-defined process and following good practices.

# References

[1] Chays, D., Shahid, J., and Frankl, P.G. Query-based test generation for database applications. In Proceedings of 1st Workshop on Testing Database Systems, 2008.

[2] GenerateData.com. A tool for generating test data. [www.generatedata.com](http://www.generatedata.com). Last accessed on February 09, 2011.

[3] SqlEdit. DTM Database Tools. [www.sqledit.com](http://www.sqledit.com). Last accessed on February 09, 2011.

[4] forSQL. forSQL Data Generator: A tool for automatics generating test data for developers. [www.forsql.com](http://www.forsql.com). Last accessed on February 09, 2011.

[5] Automation Anywhere, Inc. Automated Test Generator: A tool for generating meaningful randomized data for QA testing, load testing. [www.tethyssolutions.com/T10.htm](http://www.tethyssolutions.com/T10.htm), Last accessed on February 09, 2011.

[6] TNS Software, Inc. A tool for generating meaningful randomized data for QA testing, load testing. [www.tns-soft.com](http://www.tns-soft.com).Last accessed on February 09, 2011.

[7] Clyde, S. *CHARM* Architectural Overview. White Paper, http://charm.health.utah.gov, September 2002.

[8] Datanamic. Generating the test data from a variety of sources, including the database tables. [www.datanamic.com](http://www.datanamic.com). Last accessed on February 09, 2011.

[9] Turbo Computer Systems. TurboData: Generate the test data from a real database. [www.turbodata.ca](http://www.turbodata.ca). Last accessed on February 09, 2011.

[10] SQL Manager.net. EMS Data Generator for PostgreSQL: Generate the test data by a variety of sources. [www.sqlmanager.net/en/products/postgresql/datagenerator](file:///C:\Users\aliraza\AppData\Local\Temp\www.sqlmanager.net\en\products\postgresql\datagenerator). Last accessed on February 09, 2011.

[11] Dymek, D. and Kotulski, L. Using UML (VR) for supporting the automated test data generation. In Computational Intelligence and Security Workshops, 2007.

[12] Maddy, N.R. DBTESTGEN – Database Test Data Generator for *CHARM*. Master’s Thesis, Utah State University, 2006.

[13] Raza, A. and Clyde, S. Semantic-based test data extraction for integrated systems (*iSTDE*). In Proceedings of International Conference on Health Informatics, 2010.

[14] Narayanan A., Shmatikov V., Myths and Fallacies of “Personally Identifiable Information”, Communications of the ACM June 2010, Vol. 53, No.6

[15] Health Insurance Portability and Accountability Act of 1996. House Report 104-736.

[16] FIPS PUB 199. Standards for security categorization of federal information and Information Systems. Federal Information and Processing Standards, Feb. 2004.

[17] IBM. Optim TM Data Privacy Solution. www-01.ibm.com/software/data/optim/ core/data-privacy-solution/ Last accessed on February 09, 2011.

[18] Clyde, S. and Salkowitz, S. The Unique Records Portfolio. .Public Health Informatics Institute, 2006.

[19] Chays, D. Test Data Generation for Relational Database Applications. Doctoral Dissertation, Polytechnic University 2004.

[20] M. Dixon, J.Kohoutkova, Managing Heterogeneity in Inter-Operating Medical Information Systems, 10th ERCIM Database Research Group Workshop on Heterogeneous Information Management

[21] L. Sweeney. k-Anonymity: a model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10 (7), 2002.

[22] A. Hundepool and L. Willenborg. m- and t-argus: software for statistical disclosure control. Third International Seminar on Statistical Confidentiality. Bled: 1996.

[23] Sweeney, L. Maintaining anonymity when sharing medical data, the datafly system. MIT Artificial Intelligence Laboratory Working Paper.

[24] Narayanan A., Shmatikov V., Robust De-anonymization of Large Sparse Datasets, Proceedings of 2008 Symposium on Security and Privacy.

[25] DataMasker.com, A Net 2000 Ltd., White Paper, Data Scrambling Techniques, http://www.datamasker.com/datascramblingissues.pdf. Last accessed on March 15, 2012.