Data Scrambling Issues

A Net 2000 Ltd. White Paper

Abstract

Information in test and development databases is often masked to protect the data from inappropriate visibility. This paper provides a comprehensive overview of the issues which must be understood and dealt with in order to ensure that the information remains internally consistent yet is not inadvertently exposed in an interpretable state. A brief discussion of the motivations for masking test data and various data sanitization techniques is undertaken. The remainder of the paper is dedicated to an analysis of the many issues involved in securely scrambling data while still retaining its usability.

It must be noted that Net 2000 Ltd., (the authors of this paper), sell a software data anonymization tool called Data Masker (http://www.DataMasker.com). However, as the title states, this paper really is a generic survey of the issues associated with data scrambling and there is no further reference to any specific software. If you wish to know more, or have any questions about topics raised in this whitepaper please contact us.

Note: This paper has been superseded by the *Data Masking: What You Need To Know* white paper. We recommend you download that paper as it has expanded and enhanced content.

http://www.DataMasker.com/DataMasking_WhatYouNeedToKnow.pdf

Some keywords which may assist you in finding this document online are:

Data Sanitization, Data Sanitisation, Data Anonymization, Data Scrubbing, Data Scrambling, Data Masking, Data Obfuscation, Data Security, Data Cleansing, Data Hiding, Data Protection Act 1998, Hide Data, Disguise Data, Sanitize Data, Sanitise Data, Gramm-Leach-Bliley Act (GLBA), Data Privacy, Directive 95/46/EC of the European Parliament, Health Insurance Portability and Accountability Act (HIPAA)

Net 2000 Ltd.
http://www.Net2000Ltd.com
Info@Net2000Ltd.com

Table of Contents

Disclaimer

The contents of this document are for general information purposes only and are not intended to constitute professional advice of any description. The provision of this information does not create a business or professional services relationship. Net 2000 Ltd. makes no claim, representation, promise, undertaking or warranty regarding the accuracy, timeliness, completeness, suitability or fitness for any purpose, merchantability, up-to-datedness or any other aspect of the information contained in this paper, all of which is provided "as is" and "as available" without any warranty of any kind.

The information content of databases varies widely and each has a unique configuration. Readers should take appropriate professional advice prior to performing any actions.

Data Scrambling Issues

Why Provide Anonymous Information in Test Databases?

The information in many databases is proprietary and from a business security standpoint must be protected. If the information consists of sensitive personal details then nearly all countries place a legal obligation on the data holder which require its protection. An escape of such information can cause considerable public relations damage and have serious business or legal consequences.

So if the information is sensitive it must be protected – of that there is no doubt. The issue then becomes a matter of implementation. In general, a policy of *minimum required access* is usually adopted.

In a production environment it is usually possible to protect the information by restricting access to the underlying data. Strict controls are in place and carefully designed user interfaces present a managed view. Test and development systems are different. They present an environment in which access is usually much wider. Information is visible to more people and those people often have greater privileges and low level access. From a data visibility standpoint, test schemas have exactly the same data security requirements as production systems yet they usually contain far more relaxed controls.

In general, a reasonable security assumption is that the more people who have access to the information, the greater the inherent risk of the data being compromised. If, as is typical with test systems, it is not possible to restrict the number of people working on the data then a useful technique to provide enhanced security is to modify the data so that no sensitive information remains. The process of modifying the data to remove data sensitivity issues is known by a number of names – data masking, data sanitization, data scrubbing or data cleansing.

Irregardless of the name used, the general technique is to modify the existing data in such a way as to remove all identifiable distinguishing characteristics thus rendering the data anonymous yet still usable as a test system.

This paper will concentrate on the common issues which must be handled in order to successfully mask test data so as to render it anonymous. The methodology required is, in many cases, extremely subtle as data is often intricately interlinked and highly denormalized. Each masking operation must preserve the relationships.

Data Masking Techniques

Overview

There are a number of techniques which can be used to perform data sanitization operations on test systems.

The section below provides a quick summary of some of the more common techniques. In practice, masking the data in such a way that it remains functional requires the use of a variety of techniques – many of which are not discussed in this paper. These techniques are discussed in detail in the companion white paper available from Net 2000 Ltd. entitled Data Sanitization Techniques.

Techniques

Substitution.

This technique consists of randomly replacing the contents of a column of data with information that looks similar but is completely unrelated to the real details. For example, employee surnames are replaced with surnames drawn from a random list. Substitution is very effective in terms of preserving the look and feel of the existing data. The downside is that a largish store of substitutable information must be maintained for each column. Substitution data can sometimes be very hard to find in large quantities. For example, if a million random street addresses are required, then just obtaining the substitution data can be a major exercise in itself.

Shuffling.

A technique similar to substitution except that the substitution data is derived from the column itself. Essentially shuffling means the data in a column is randomly moved between rows until there is no longer any reasonable correlation with the remaining information in the row. There are certain dangers inherent to the shuffling technique – see the discussion below regarding the *Isolated Case* and *Meta Information* issues. Shuffling is rarely effective when used on small amounts of data. For example, if there are only 5 rows in a table it probably will not be too difficult to figure out which item of shuffled data really belongs to which row.

Number and Date Variance.

The Variance technique is useful on numeric and date columns. Simply put, the algorithm involves modifying each value in a column by some random percentage of its real value. This technique has the nice advantage of providing a reasonable disguise for the exact details while still keeping the range and distribution of values in the column within viable limits. For example, a column of birth dates might have a random variance of 10% placed on it. Some values would be higher, some lower but all would not be too far from their original range.

Data Scrambling Issues

Overview

The previous discussion of Data Masking techniques might leave the impression that a quick substitution operation or shuffling is all that is required in order to render test data anonymous. This is far from the case – if care is not taken the data may be masked but the subsequent database could be unusable or insecure in subtle ways. Consideration must be given to a number of issues in order for the data to be successfully sanitized.

The following sections list a number of identifiable data scrambling issues. The names used are ours and were composed in order to provide a short descriptor for later reference. As far as we know there is no widely agreed nomenclature.

Issues

Relevant Data.

In test systems, most data will eventually appear on a front end screen in one form or another. To be useful, the sanitised values must resemble the look-and-feel of the original information. For example, surnames should be replaced with random surnames. Usually it is not acceptable to insert random collections of meaningless text.

Row-Internal Data Synchronization.

In many cases the contents of one column in a row are related in some way to the contents of the other columns in the same row. For example, if the gender field in the row is `‘F’` then the `FIRST_NAME` column should really have a female first name. Sometimes a column contains duplicate (denormalized) or aggregate information from the other columns in the row. In this case, the column should reflect the equivalent details after anonymization. An example of this issue is the commonly seen `FULL_NAME` column used in many systems.

Table-Internal Data Synchronization.

This issue is quite common in many systems. The rows in the table are massively denormalized and contain information that must be identical among many rows. For example, in an Oracle HR schema in the `PER_ALL_PEOPLE_F` table each `PERSON_ID` can have multiple rows (one for every assignment that person has ever had). Each row has a `LAST_NAME` field. When sanitizing data, every distinct `PERSON_ID` must be updated with the same anonymous `LAST_NAME`. If this is not done, an employees name will appear on the front end screens as having changed dozens of times.

Table-Table Data Synchronization.

Quite often in test systems, important information which must be obfuscated is used as a join key to a column in one (or more) other tables. This means data masked in one table must have synchronized data changes in a number of others. For example, changes to the `EMPLOYEE_NUMBER` column in one table must trigger identical changes in other tables. Quite often the target column requiring synchronization will not have the same name. For example, in Oracle HR systems the `ASSIGNMENT_ID` of the `PER_ALL_ASSIGNMENTS_F` table must be updated to be identical to the `EMPLOYEE_NUMBER` column.

Multi-Synchronization.

It is possible for a column of a table to be constrained by multiple simultaneous synchronization issues. The `EMPLOYEE_NUMBER` of the `PER_ALL_PEOPLE_F` table, in Oracle HR schemas has both *Table-Internal* and *Table-Table Synchronization* issues.

Intelligent Keys.

It often happens that data items have a structure which represents an internal meaning. An example of this is the checksum on a credit card number. It is undesirable to sanitise such data by replacing it with a random collection of digits. The problem arises in the front end screens – they check the content prior to update. Hence if the data value is not right according to its internal design, any screen which displays the value will never permit an update because the validity checks fail. Intelligent Keys are commonly found in such things as employee numbers, credit card numbers and ID numbers. There are two ways to resolve this issue – either generate data items that meet the validity standard or shuffle the data in the column among the rows so that no row contains its original data but each data item is valid internally. It is a matter of judgement whether shuffling the column data among the rows provides sufficient sanitization for the data.

Free Format Data.

Textual data such as letters, memos, disciplinary notes etc are practically impossible to sanitize in-situ. Unless the masking algorithms are extremely clever, or the format of the text is fixed it is probable that some information will be missed during the sanitization process. The usual way of dealing with free format data is to replace all values with randomly generated meaningless text (or simply null them) and then update certain selected data items with carefully hand sanitized examples. This will give the users of the test system some realistic looking information to work on while preserving anonymity in the remainder.

Field Overflow.

Care must be taken when replacing the real data with false test data to avoid overflowing previously allocated storage capacity. For example, if a field being masked is 20 characters in size then no items of greater length can be used as replacement data otherwise errors will be generated and the process will fail.

Consistent Masking.

A common requirement for a sanitization process is to ensure that the output is consistent across multiple runs. In practice this means that if the name of employee Joe Smith gets changed to Bill Jones then the next time the database is cloned and sanitized Joe Smith should again appear as Bill Jones (not Jim Williams). Training teams, in particular, tend to require this feature as they use a lot of pre-scripted examples.

Isolated Case Phenomena.

The end result of the sanitization on the test and development database is to preserve the privacy of the individual records. In general, anonymity is derived from the presence of a large number of similar records. If a record stands out in any way it could be attributable to an individual. For example, could the record for the organizational CEO be determined by finding the largest salary in the table? Sometimes each record is its own special case. An unmasked birth date could readily be attributable to a specific individual. Whether this issue is important depends largely on how the remainder of the information is masked.

Meta Information.

Sometimes even if information is not attributable to a specific individual, the collection of information might well be sensitive. For example, salary figures may be anonymous because the associated employee names have been masked, but does it matter if someone can add up the salary figures for a department? It's a judgement which has to be made considering the specific circumstances.

WHERE Clause Skips.

When conducting masking operations be careful how the data to be sanitized is selected with `WHERE` clauses. It is easy, by making assumptions about the content of data in the row, to leave data in some rows in its original state. As an example, consider a table with a `FIRST_NAME` column and a `GENDER` column. Don't replace all the `FIRST_NAME` fields where `GENDER='M'` with male first names and the `FIRST_NAME` fields where `GENDER='F'` with female first names unless you are absolutely sure that the `GENDER` column can contain only `'M'` or `'F'`. It is entirely possible that the `GENDER` field may contain some other character (including null). Masking only the `'M'` and `'F' GENDER` fields will leave the `FIRST_NAME` field in some rows unmasked. It is far better to mask all rows with one option (Male First Names for example) and then go through a second time to mask every `FIRST_NAME` fields where `GENDER='F'` with female first names. This ensures that all rows have some sort of masking operation applied – irregardless of the state of the `GENDER` field. Where Clause Skips can lead to some quite insidious omissions – be sure to use full coverage to ensure every record gets masked.

Granularity.

Is it necessary to sanitize absolutely everything? Or is masking enough data to prevent attribution sufficient. For example, do job titles have to be masked? Perhaps just removing a few examples of the *Isolated Case Phenomena* is sufficient. Either way, decisions have to be made as to the depth of cleansing required. Any sanitization process trades thoroughness for complexity – the deeper operation the harder it is to maintain synchronization.

Distribution Preservation.

In many cases the distribution of the data (numbers and dates) is important. For example, if salary figures are randomly updated, the random number generator will almost certainly give an even distribution between the specified ranges rather than the usual pyramid of lots of small salaries and fewer larger ones. Whether the skewing of the data matters is an implementation decision. If the data distribution is important then the *Variance* techniques previously discussed are the tool to use.

Sparse Data.

Not all columns in a table have data in all rows. For example, the `PREVIOUS_LAST_NAME` field in the `PER_ALL_PEOPLE_F` table will be mostly empty. In this case, in order to preserve usability, it is not appropriate to fill in every `PREVIOUS_LAST_NAME` – the majority must remain null.

Percentage Operations.

The Sparse Data issue above highlighted the point that not every column in a table might necessarily have data. The column sparseness can be preserved by masking the not null values. However, it might be desirable to assign the `PREVIOUS_LAST_NAME` values randomly thus removing that association from any specific row. In cases like this, it is useful to be able to null everything and then randomly update just a specified percentage of the rows with the appropriate contents.

Sequential Operations.

The order in which masking operations are applied is sometimes very important. For example, in the *Row Internal Synchronization* issue discussed above, a `FULL_NAME` field was built from various other columns in the row (`LAST_NAME`, `FIRST_NAME` etc). It is essential that the build of the `FULL_NAME` field is applied after all of the masking rules for each component have successfully completed. In general, it cannot be assumed that a sequence of masking operations are separable. Not only does each masking operation have to complete - they sometimes have to complete in a specific order.

Special Cases.

Most masking operations sweep with a broad brush and tend to obliterate special cases. For example, if there are only a few examples of an employee that has quit and then been rehired it may well prove to be the case that they received the same `PERSON_ID` but a different `EMPLOYEE_NUMBER` when hired the second time. The *Table-Internal Synchronization* operation may well remove this special case. Whether this homogenisation of the information is important is a decision for each implementation – however it is useful to realize that the issue exists.

User Defined Fields (Flex Fields).

Many vendors of pre-prepared software packages assume (quite rightly) that every site will have custom requirements for the storage of information and implement user defined fields for this purpose. These fields store site specific information and their usage varies widely between implementations. An analysis of the user defined field contents is required in order to ensure the data is completely scrubbed clean of personal details.

Speed.

Some of the tables are big. One has to be careful how the masking operation is performed otherwise it will take an inordinate amount of time to complete. For example, it is not possible to perform Inter-Table Data Synchronization directly on an un-indexed column in a large table. The process just never finishes. That is not to say it is impossible to do – one just has to use a carefully constructed intermediate table and a clever update statement. In test systems, make sure to disable all triggers before performing any updates or even operations on small tables will take forever to complete.

Repeatability.

It is probable that any sanitization operation on a test system will need to be reproduced many times in the future as each new test instance is cloned. Make sure the masking process is designed to be simple and repeatable.

Encryption.

Often when considering a data masking approach, the idea of encryption is raised. Although there may be specific cases where encryption is a desirable option, in general, encryption of the test data is not a very useful data sanitization strategy. Any reasonable encryption mechanism will convert the data into binary characters which then renders the data unusable from a test and development point of view. Front end screens and printers do not cope well with binary data and the information does not look realistic. Also there is no guarantee that the encrypted data will be of an identical size leading to field overflow problems. In addition, and possibly the most serious from a security standpoint, encrypting test data to mask it leads to key management issues. Data is encrypted using a key – if the key escapes then every test system encrypted with that key is then compromised. Other methods do not have that issue – usually there is no way to re-generate the original data from the scrambled system.

Encryption II (Caesarean Ciphers).

In order to “get around” the binary data output issues associated with masking by encryption one occasionally observes test systems masked using a mechanism known as a Caesarean Cipher. This is the technique, (the letter ‘A’ is replaced by ‘C’, ‘B’ is replaced by ‘D’ etc), that most people think of when first approaching an encryption operation. It is easy to implement, very fast and seems to do the job. However, it is appallingly insecure – never use this method to encrypt any data. It is a trivial matter for any sufficiently motivated person to write a program to decrypt such data based on letter frequency appearances. In fact, first year computer science students are often asked to write such programs as assignments.

Summary

Given the legal and organizational operating environment of today, many test and development databases will require some form of sanitization in order to render the informational content anonymous.

There are a variety of techniques available, and an even larger number of issues of which to be aware. A companion white paper which discusses these techniques in detail can be downloaded via the URL
http://www.DataMasker.com/datasanitization_whitepaper.pdf

A paper dealing with the specifics of rendering Oracle Apps HR data anonymous can be found at the URL:
http://www.DataMasker.com/AnonymousOracleAppsHRData.pdf

It is important when sanitizing data in test systems that a number of critical issues are considered. If these issues are not dealt with then the resulting database may be rendered unusable or could be insecure.