
Data Masking: What You Need to Know

What You Really Need To Know Before You Begin

A Net 2000 Ltd. White Paper

Abstract

It is often necessary to anonymize data in test and development databases in order to protect it from inappropriate visibility. There are many things, some incredibly subtle, which can cause problems when masking data. This paper provides a survey of the practical issues involved in the masking of sensitive data and tells you the things you really need to know about before getting started.

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 knowledge you really need to have before getting involved in the masking of data and there will be no further reference to any specific software. If you wish to know more, or have any questions about issues raised in this whitepaper please contact us.

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), GDPR, General Data Protection Regulation

Net 2000 Ltd.

<http://www.Net2000Ltd.com>

Info@Net2000Ltd.com

Table of Contents

Disclaimer	1
What Does Data Masking Mean?	2
Why Mask Data?.....	3
Legal Requirements	3
Loss of Confidence And Public Relations Disasters	3
Malicious Exposure	3
Accidental Exposure	3
What Data To Mask	4
Use A Variety Of Masking Routines	4
Light Masking on a Bug-Fix or Fire-Fighting Database	4
Medium Masking on Internal Development Databases	4
Thorough Masking on an Outsourced Database	4
Data Masking Architectures	5
On the Fly, Server-To-Server, Data Masking Architectures	5
In-Situ Data Masking Architectures	5
Data Masking Techniques.....	6
Substitution	6
Shuffling	6
Number and Date Variance.....	7
Encryption.....	7
Nulling Out/Truncating.....	8
Masking Out Data	8
Row Internal Synchronization	8
Table Internal Synchronization.....	9
Table-To-Table Synchronization	10
Table-To-Table Synchronization On Primary Key.....	11
Table-To-Table Synchronization Via Third Table	11
Synchronizing Between Different Datatypes.....	12
Cross Schema Synchronization.....	13
Cross Database Synchronization.....	13
Cross Server Synchronization.....	13

Cross Platform Server Synchronization	13
Selective Masking: Ability to Apply a WHERE Clause.....	14
Selective Masking: Ability to Apply Sampling	14
User Defined SQL Commands	14
Flat File Masking	14
Multi-threading and Internal Scheduling	15
Rule Parallelism	15
Sequences Within the Parallel Runs	15
Data Masking Issues	16
Where Clause Skips	16
Table-To-Table Skips	16
Cleaning up the Data.....	17
Isolated Case Phenomena.....	17
Relevant Data.....	17
Intelligent Keys	17
Free Format Data	18
Field Overflow	18
Sparse Data	18
Percentage Operations.....	18
Sequence Generation.....	19
Consistent Masking.....	19
Aggregate Information.....	19
Meta Information	19
Granularity	19
Distribution Preservation	20
Special Cases	20
User Defined Fields	20
Speed.....	20
Repeatability	21
Summary	22

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 Masking: What You Need to Know

What Does Data Masking Mean?

Data Masking is the replacement of existing sensitive information in test or development databases with information that looks real but is of no use to anyone who might wish to misuse it. In general, the users of the test, development or training databases do not need to see the actual information as long as what they are looking at looks real and is consistent.

The ability of test and development teams to use masked data is not universally true. The *What Data To Mask* section in this paper addresses the issues associated with this and presents options which can be used to mitigate the data exposure risk.

It is important to be aware that data masking is appropriate to more than just personal details – sometimes business confidential information is appropriate for masking as well. For example, it may be desirable to prevent quarterly sales figures for some products being present in an outsourced test database.

Data masking is not the same thing as restricting the visibility of information in production databases from people who are not authorized to see it. In that situation, the data is actually present in the database and is simply not visible to the unauthorized. There are many good and justifiable reasons for taking this approach in a production system, but adopting a “*data is present but hidden*” approach to the protection of data in test and development databases is a recipe for trouble. The reason is that strict controls are in place in production databases and these can present a carefully managed view. Test and development systems are different. Typically, they are 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, a test or dev system in which the data is present but hidden is a system which sooner or later will expose its data.

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. The modification of the existing data in such a way as to remove all identifiable distinguishing characteristics yet still usable as a test system can provide a valuable layer of security for test and development databases.

Why Mask Data?

Legal Requirements

The regulatory environment surrounding the duties and obligations of a data holder to protect the information they maintain are becoming increasingly rigorous in just about every legal jurisdiction. It is a pretty safe assumption that the standards for the security and maintenance of data will become increasingly strict in the future.

Loss of Confidence And Public Relations Disasters

It can reasonably be said in most locations, that if a data escape happens at your organization, then the formal legal sanctions applied by governmental bodies is not the only problem you will be facing. Possibly it may not even be the biggest of your immediate worries.

Inappropriate data exposure, whether accidental or malicious, can have devastating consequences. Often the costs of such an event, both actual and un-quantifiable can far exceed any fines levied for the violation of the rules. For example, what will it cost the organization if potential customers are not willing to provide sensitive information to your company because they read an article about a data escape in the newspaper. Dealing with the public relations aftermath of seeing the companies name in the press will not be cheap. It also does not take much imagination to realize that senior management are not going to be happy about having to give a press conference to reassure the public. The public relations costs of a data escape usually far exceed the sanctions levied by governmental organizations.

Malicious Exposure

Most people think the major risk to the information they hold is external entities (and organized syndicates) out to break in and steal the data. The assumption then follows that protecting the network and firewalls is the appropriate and sufficient response. There is no denying that such protection is necessary – however it has been shown that in many cases the data is stolen by malicious insiders who have been granted access to the data. No firewall can keep an insider from acquiring data under such circumstances. However, by reducing the number of databases with unmasked information, the overall risk of exposure is mitigated. The external hackers, if they get through the network security, will have far fewer useable targets and a far greater proportion of the inside personnel will have no access to the real data.

Accidental Exposure

The risk of accidental exposure of information is often neglected when considering the security risks associated with real test data. Often it is thought that “*there is no point in masking the test data because everybody has access to production anyways*”. Not so, the risks associated with an accidental exposure of the data remain. Often just masking the most sensitive information (credit card numbers, customer email addresses etc) is enough to somewhat mitigate the damage associated with accidental exposure and the masked databases remain just as functional.

What Data To Mask

Use A Variety Of Masking Routines

One of the common mis-perceptions regarding data masking is the idea that “*our data is not suitable for anonymization because the real information is needed in some of the test databases*”. The actual data may well be required in one database – but it is unlikely to be needed in every test database. A reasonable course of action is to have a variety of masking routines for different purposes. These masking routines would be tuned to the degree of exposure of the data and the amount of control maintained.

Light Masking on a Bug-Fix or Fire-Fighting Database

One of the major issues associated with data masking (see the Data Masking Issues section below) is that the masking process can sometimes “tidy up” the data. In order to be effective, a Bug-Fix or Fire-Fighting database needs to have as few changes as possible. However, there are a number of items which can be safely masked - even in a Bug-Fix database. Things like bank account or credit card numbers, unless they are used as join keys, can usually be masked to provide some protection. In general, any opaque information which is only meaningful to an external organization can be masked in these circumstances.

Medium Masking on Internal Development Databases

Databases which are used by internal development, test and training staff and have no visibility outside the organization should probably receive a medium level of masking. In general, it is unwise to assume that “*since everybody with access to the test databases also has access to the production database there is no need to mask the data*”. Accidents happen, and you will need to protect yourself from that. The *Accidental Exposure* section above discusses this issue in detail. Masking the most common personally identifiable information in these databases as well as the sensitive details such as bank account numbers is usually sufficient.

Thorough Masking on an Outsourced Database

If operational control of the test and development databases will be handed over to a third party then a strong case for a through anonymization of the contents can be made. A database going offsite to an outsourced development team might have an extremely thorough masking applied and only the real information absolutely necessary to enable the remote personnel to perform their function would be present.

Data Masking Architectures

Fundamentally, there are two basic types of architectures which are used in the design of data masking software.

On the Fly, Server-To-Server, Data Masking Architectures

In this architecture the data does not exist in the target database prior to masking. The anonymization rules are applied as part of the process of moving the data from the source to the target. Often this type of masking is integrated into the cloning process which creates the target database.

Advantages

- The data is never present in an unmasked form in the target database.

Disadvantages

- Any errors in the process necessarily interrupt the transfer of the data.
- The ability to mask data after the transfer has completed can be troublesome. This might happen in cases where the masked target database has been built and it is subsequently decided that a specific column of information really needs to be masked. In this case, the masking software needs to have In-Situ masking capabilities (see below) or the entire clone and masking operation will need to be repeated.
- The ability to use alternative, perhaps preferred, tools to perform the cloning operation is impacted.

In-Situ Data Masking Architectures

In this style, the clone of the database to be masked is created by other means and the software simply operates on the cloned database. There are two types of in-situ masking: masking rules which are executed and controlled as a standalone entity on the target and data masking rules which are controlled by a different system which then connects to the target and controls the execution of the rules.

Advantages

- It is possible to apply additional masking operations at any time.
- The masking operations are separate from the copy process so existing cloning solutions can be used and the data masking rules are possibly simpler to maintain.

Disadvantages

- The data is present in an unmasked state in the target database and hence additional security measures will be required during that time.

Data Masking 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, the surnames in a customer database could be sanitized by replacing the real last names with surnames drawn from a largish 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 available for each column to be substituted. For example, to sanitize surnames by substitution, a list of random last names must be available. Then to sanitize telephone numbers, a list of phone numbers must be available. Frequently, the ability to generate known invalid data (credit card numbers that will pass the checksum tests but never work) is a nice-to-have feature.

Substitution data can sometimes be very hard to find in large quantities - however any data masking software should contain datasets of commonly required items. When evaluating data masking software the size, scope and variety of the datasets should be considered. Another useful feature to look for is the ability to build your own custom datasets and add them for use in the masking rules.

Shuffling

Shuffling is similar to substitution except that the substitution data is derived from the column itself. Essentially 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 is a certain danger in the shuffling technique. It does not prevent people from asking questions like *"I wonder if so-and-so is on the supplier list?"* In other words, the original data is still present and sometimes meaningful questions can still be asked of it. Another consideration is the algorithm used to shuffle the data. If the shuffling method can be determined, then the data can be easily "un-shuffled". For example, if the shuffle algorithm simply ran down the table swapping the column data in between every group of two rows it would not take much work from an interested party to revert things to their un-shuffled state.

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 of the shuffled data really belongs to which row. On the other hand, if a column of numeric data is shuffled, the sum and average of the column still work out to the same amount. This can sometimes be useful.

Shuffle rules are best used on large tables and leave the look and feel of the data intact. They are fast, but great care must be taken to use a sophisticated algorithm to randomise the shuffling of the rows.

Number and Date Variance

The Number Variance technique is useful on numeric or date data. Simply put, the algorithm involves modifying each number or date 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 data while still keeping the range and distribution of values in the column to within existing limits. For example, a column of salary details might have a random variance of $\pm 10\%$ placed on it. Some values would be higher, some lower but all would be not too far from their original range. Date fields are also a good candidate for variance techniques. Birth dates, for example, could be varied with in an arbitrary range of ± 120 days which effectively disguises the personally identifiable information while still preserving the distribution.

The variance technique can prevent attempts to discover true records using known date data or the exposure of sensitive numeric or date data.

Encryption

This technique offers the option of leaving the data in place and visible to those with the appropriate key while remaining effectively useless to anybody without the key. This would seem to be a very good option – yet, for anonymous test databases, it is one of the least useful techniques.

The advantage of having the real data available to anybody with the key – is actually a major disadvantage in a test or development database. The “optional” visibility provides no major advantage in a test system and the encryption password only needs to escape once and all of the data is compromised. Of course, you can change the key and regenerate the test instances – but outsourced, stored or saved copies of the data are all still available under the old password.

Encryption also destroys the formatting and look and feel of the data. Encrypted data rarely looks meaningful, in fact, it usually looks like binary data. This sometimes leads to character set issues when manipulating encrypted varchar fields. Certain types of encryption impose constraints on the data format as well. In effect, this means that the fields must be extended with a suitable padding character which must then be stripped off at decryption time.

The strength of the encryption is also an issue. Some encryption is more secure than others. According to the experts, most encryption systems can be broken – it is just a matter of time and effort. In other words, not very much will keep the national security agencies of largish countries from reading your files should they choose to do so. This may not be a big worry if the requirement is to protect proprietary business information. Never, ever, use a simplistic encryption scheme designed by amateurs. For example, one in which the letter ‘A’ is replaced by ‘X’ and the letter ‘B’ by ‘M’ etc. is trivially easy to decrypt based on letter frequency probabilities. In fact, first year computer science students are often asked to write such programs as assignments.

Nulling Out/Truncating

Simply deleting a column of data by replacing it with NULL values is an effective way of ensuring that it is not inappropriately visible in test environments. Unfortunately it is also one of the least desirable options from a test database standpoint. Usually the test teams need to work on the data or at least a realistic approximation of it. For example, it is very hard to write and test customer account maintenance forms if the customer name, address and contact details are all NULL values. NULL'ing or truncating data is useful in circumstances where the data is simply not required, but is rarely useful as the entire data sanitization strategy.

Masking Out Data

Masking data, besides being the generic term for the process of data anonymization, means replacing certain fields with a mask character (such as an X). This effectively disguises the data content while preserving the same formatting on front end screens and reports. For example, a column of credit card numbers might look like:

```
4346 6454 0020 5379
4493 9238 7315 5787
4297 8296 7496 8724
```

and after the masking operation the information would appear as:

```
4346 XXXX XXXX 5379
4493 XXXX XXXX 5787
4297 XXXX XXXX 8724
```

The masking characters effectively remove much of the sensitive content from the record while still preserving the look and feel. Take care to ensure that enough of the data is masked to preserve security. It would not be hard to regenerate the original credit card number from a masking operation such as: 4297 8296 7496 87XX since the numbers are generated with a specific and well known checksum algorithm. Also care must be taken not to mask out potentially required information. A masking operation such as XXXX XXXX XXXX 5379 would strip the card issuer details from the credit card number. This may, or may not, be desirable.

If the data is in a specific, invariable format, then Masking Out is a powerful and fast option. If numerous special cases must be dealt with then masking can be slow, extremely complex to administer and can potentially leave some data items inappropriately masked.

Row Internal Synchronization

Data is rarely consistently available in a fully normalized format. Consider the example below in which the FULL_NAME field is composed of data from other columns in the same row.

	PERSON_ID	EMP_NO	FIRST_NAME	LAST_NAME	FULL_NAME	DATA
1	100	1000777	Robert	Smith	Robert Smith	A67e
2	200	1000888	Sue	Jones	Sue Jones	B363
3	300	1000999	Thomas	Wells	Thomas Wells	C678

After the data has been masked, the `FIRST_NAME` and `LAST_NAME` columns will have been changed to other values. For the information to be secure, clearly the `FULL_NAME` field must also change. However, it must change to contain values synchronized with the rest of the data in the row so that the masked data reflects the denormalized structure of the row. This type of synchronization is called Row-Internal Synchronization and it is quite distinct from the other two types: Table-Internal and Table-To-Table Synchronization.

The Row-Internal Synchronization technique updates a field in a row with a combination of values from the same row. This means that if, after masking, the `FIRST_NAME` and `LAST_NAME` change to Albert and Wilson then (in this example) the `FULL_NAME` column should be updated to contain Albert Wilson. Row-Internal Synchronization is a common requirement and the data scrambling software you choose should support it.

Table Internal Synchronization

Sometimes the same data appears in multiple rows within the same table. In the example below, the name Robert Smith appears in the `FIRST_NAME` and `LAST_NAME` columns in multiple rows.

	PERSON_ID	EMP_NO	FIRST_NAME	LAST_NAME
1	100	1000777	Robert	Smith
2	200	1000888	Sue	Jones
3	300	1000999	Thomas	Wells
5	100	1000777	Robert	Smith
6	100	1000777	Robert	Smith
7	300	1000999	Thomas	Wells
8	300	1000999	Thomas	Wells
9	200	1000888	Sue	Jones

In other words, some of the data items are denormalized because of repetitions in multiple rows. If the name Robert Smith changes to Albert Wilson after masking, then the same Robert Smith referenced in other rows must also change to Albert Wilson in a consistent manner. This requirement is necessary to preserve the relationships between the data rows and is called Table-Internal Synchronization.

A Table-Internal Synchronization operation will update columns in groups of rows within a table to contain identical values. This means that every occurrence of Robert

Smith in the table will contain Albert Wilson. Good data anonymization software should provide support for this requirement.

Table-To-Table Synchronization

It is often the case in practical information systems that identical information is distributed among multiple tables in a denormalized format. For example, an employee name may be held in several tables. It is desirable (often essential) that if the name is masked in one column then the other tables in which the information is held should also be updated with an identical value. This requirement is called Table-To-Table Synchronization and is the type of synchronization most people think of when considering a data anonymization process. It should be noted that there are also two other types of synchronization: Row-Internal and Table-Internal, (see above) and all three have quite different purposes and functions.

Table-To-Table Synchronization operations are designed to correlate any data changes made to a table column with columns in other tables. Thus performing Table-To-Table Synchronization operations requires the knowledge of a join condition between the two tables. The join condition is used to ensure the appropriate rows are updated correctly. Also required, is knowledge of the columns in each of the source and target tables which must be synchronized.

As an example, consider the two tables:

```
TABLE SOURCE
  COLUMN IDNUM number(10)
  COLUMN NAME varchar(40)
```

```
TABLE TARGET
  COLUMN IDNUM number(10)
  COLUMN NAME varchar(40)
```

Assume the NAME column must be synchronized between the two tables and the join condition is SOURCE.IDNUM=TARGET.IDNUM. If a Substitution operation is performed on the SOURCE.NAME column and a subsequent Table-To-Table Synchronization operation is applied, then each TARGET.NAME column will receive values identical to the ones in the SOURCE table where the join condition matches.

Important Note: It is important to realize that if there are IDNUM values in the TARGET table that are not present in the SOURCE table, the Table-To-Table Synchronization rule will have no effect on those rows. This issue is called a *Table-To-Table Skip* and is discussed in detail in the Data Masking Issues section.

Table-To-Table Synchronization is probably the most common synchronization task – in fact, it is rare that a data scrambling process does not require it. Every data masking solution should provide support for this operation.

Table-To-Table Synchronization On Primary Key

There are some special cases of Table-To-Table Synchronization which deserve their own discussion. The first is a scenario in which the masked data must be synchronized between tables and in which the data being masked also forms the join condition between the tables.

Below is an example of a table (we will refer to this table as the PARENT table) and its contents prior to any masking operation. Notice how the CUSTOMER_ID field contains the first 5 letters of the customer name. It might be necessary to mask the CUSTOMER_ID value in order to ensure the anonymity of the database.

```
CUSTOMER_ID COMPANY
SMITH000001 ABC Co.
BROWN000002 DEF Ltd.
ARMST000005 HIJ LLC.
WHITE000007 LMN Partners
```

The problem lies in the fact that the CUSTOMER_ID value is also used in one or more other tables. The table below illustrates this (we will refer to this as the CHILD table).

```
CUSTOMER_ID CUSTOMER_NOTES
SMITH000001 Very nice customer.
BROWN000002 Courier always charges extra shipping
ARMST000005 This customer is actually a PO box - take care.
WHITE000007 Has a phobia about dogs - never mention them.
```

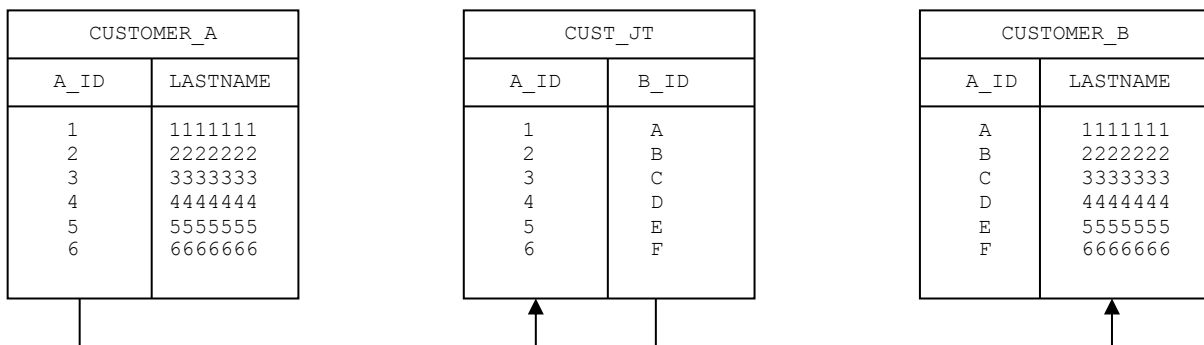
If masking operations change the CUSTOMER_ID for SMITH000001 to JONES000099 in the PARENT table, then the SMITH000001 value in the CHILD table must also change to JONES000099. Such synchronization would normally be implemented with a standard Table-To-Table Synchronization operation except for the fact that the CUSTOMER_ID value is the join relationship between the two tables. The instant the CUSTOMER_ID in the PARENT table is masked to some other value, the relationship is destroyed and the CHILD table rows can no longer be associated with their original PARENT rows. The two CUSTOMER_ID values are no longer equal and the join condition no longer exists.

Any data anonymization solution which is implemented should be able to handle the special case requirement of Table-To-Table masking on join key without much manual intervention.

Table-To-Table Synchronization Via Third Table

The second special case in Table-To-Table Synchronization operations is a scenario in which the two tables have no direct join relationship. In other words, the data columns must be synchronized between the two tables but the equivalent rows must be identified by a join made through a third table.

Below is an example of a situation in which the LASTNAME data in the CUSTOMER_B table is identical to that in the LASTNAME column of the CUSTOMER_A table. The table CUST_JT is the join table which describes how the rows in the two tables are related.



In the above diagram, the rows in the CUSTOMER_B table are not directly related to the rows in the CUSTOMER_A table. Equivalent rows in the CUSTOMER_A and CUSTOMER_B tables are identified by joining through the CUST_JT table.

If masking operations change the LASTNAME in the CUSTOMER_A table, then table-to-table synchronization operations which update the equivalent rows in the CUSTOMER_B table to contain identical values will need to be performed.

This requirement is much less common than the other synchronization techniques (discussed above) but it is advisable to be aware that it may exist. There are a variety of ways of performing this operation – the authors of this paper have issued a generic technical note on the subject.

Synchronizing Between Different Datatypes

In many cases, a set of data will contain the same data entity in multiple locations but this data will be stored in different formats. For example, a CUSTOMER_ID field might have a datatype of VARCHAR in one table and INTEGER in another. If this is the case, problems can arise from two sources: if the values from one column are substituted into the other as part of a synchronization operation or if the columns are used as join keys in order to synchronize other data.

Taking the first case in which the values from one column are copied to another, it can be seen that problems can arise both from the fact that a conversion operation needs to be performed before the update is possible and also from the fact that the source column might contain data unsuitable for conversion. In the above example, the synchronization routines would need to be able to convert numbers stored as characters into integer values in order to update that column. In some situations this conversion will be done automatically by the database and in some cases it needs to be explicitly specified by the designer of the masking routines. If there is a, perhaps erroneous, CUSTOMER_ID value of “!0001a” in the source table it will not be possible to cast this value to an integer. In this event, the designer of the masking operations will need to be aware that this event will happen and implement a handler for that special case.

If the data items are used as join keys for other columns, then the issue of equivalency detection presents itself. For example, if a column of last names in two tables is to be synchronized and the rows are related by a `CUSTOMER_ID` which is a `VARCHAR` in one table and an `INTEGER` in another then the update statement must be able to cope with this. In effect, it must be able to detect that a value of "10001" as a character string is the same thing as the number 10001. This is not super difficult in theory – however the data masking system implemented must be prepared to cope with it. If the amount of data to be compared is large then the efficiency of this comparison also plays a major factor in the execution time of the masking operation.

Cross Schema Synchronization

Many databases contain de-normalized data in tables which are located in multiple schemas. If this data is related, a Table-To-Table Synchronization operation may be required after the data masking operations have concluded. The analysis phase conducted before the construction of the masking routines should pay attention to this requirement and the masking software should be able to support it if required.

Cross Database Synchronization

Similar to the requirement for Cross Schema Synchronization is the requirement for Cross Database Synchronization. In this particular scenario the databases are co-located in the same server but the masked data is located in separate databases (and also schemas). If this requirement exists, the analysis phase should carefully plan for it and the database software should be able to support such a synchronization operation.

Cross Server Synchronization

As with Cross Schema and Cross Database Synchronization it is sometimes necessary to synchronize data between platforms. This type of synchronization necessarily involves the physical movement of the data between distinct environments and hence presents a certain technical challenge if the data volumes are large. Fundamentally there are two approaches to the movement of the data: one can use the inter-database transport mechanisms supplied natively by the database vendor, or the data movement can be handled and copied by the data masking software. Typically the inter-database data transportation mechanisms are highly optimised by the database vendor and also often have advanced functionality such as the ability to create and replicate an entire table as a single command.

Cross Platform Server Synchronization

The scenario where data must be kept consistent between tables which are located in physically different servers in databases from different vendors is probably the hardest type of synchronization operation to perform. Many vendors supply data transportation compatibility routines which can move the data to and from servers from other vendors. If this type of synchronization is required, then the data masking software must have the ability to support the use of these routines or have the ability to perform the cross platform copy operation itself.

Selective Masking: Ability to Apply a WHERE Clause

It is essential to be able to use specific criteria to choose the rows on which the masking operations are performed. In effect, this means that it must be possible to apply a *Where Clause* to a set of data and have the masking operations apply only to that subset of the table. As an example of a *Where Clause* requirement, consider a masking operation on a column containing first names. These names are gender specific and the end users of the database may well require Male and Female names to be present in the appropriate rows after the masking operations complete. Two rules, each with a *Where Clause* based on the gender column will be required here. There is a potential trap here – note the discussion entitled *Where Clause Skips* in the Data Masking Issues section of this document.

Selective Masking: Ability to Apply Sampling

It is often useful to be able to have the ability to apply masking operations to a sample of the data in a table. For example, a column of surnames might be masked using a large dataset of surnames. In reality, some names are much more common than others. In order to ensure that enough duplicate names are present and the masked data better represents reality, it may be desirable to apply a subsequent masking rule to 10 or 20 percent of the rows in the table using a small dataset of common last names.

If sampling is used, it is important that a sophisticated sampling algorithm is used. If 25 percent of a table is to be masked then it is usually undesirable to mask only the top quarter of the rows in the table or exactly every fourth row. The sampled rows should be retrieved randomly from the entire contents of the table.

User Defined SQL Commands

It is very helpful, essential in many circumstances, to be able to run user defined SQL statements. Such statements could be used, for example, to create an index which speeds up the operation of other rules or to assist with a complex synchronization operation.

Note that many databases use different internal mechanisms for the creation and execution of a block of statements (i.e. a procedure) than they do for simple SQL statements. It is usually important that the solution chosen be able to support both block SQL constructs as well as simple insert, update and delete statements.

Flat File Masking

Flat files of data are not as common as they used to be – however they still exist. If your organization still uses flat files (perhaps for data transfer) and they actually require anonymization, then support for masking operations on those files will be needed in your masking solution.

Multi-threading and Internal Scheduling

Rule Parallelism

Many databases contain extremely large tables. If hundreds of millions of rows must be masked, the anonymization operations can take a considerable amount of time.

For large databases it is very useful to be able to run multiple masking operations simultaneously. Fundamentally there are two approaches to the parallelization requirement. In the first approach masking operations are executed in parallel but only against distinct tables. In other words, no table can have simultaneous masking operations applied to it. This approach solves a number of the locking issues which can occur when large updates are applied to the same table – however in many databases there are relatively few large tables and hence little opportunity to run the masking operations in parallel. The second, more sophisticated, approach permits multiple simultaneous data anonymization operations against the same table but these operations are designed to work with the database locking mechanism in such a way that lock collisions either do not occur or are resolved transparently without any operator intervention.

Sequences Within the Parallel Runs

Not all rules can execute simultaneously – either within the same table or on different tables. As an example consider a simple masking operation involving Row Internal Synchronization. This type of synchronization is discussed in detail in the Data Masking Techniques section. In the example below if the `FIRST_NAME` column and `LAST_NAME` column are masked then the `FULL_NAME` column must be rebuilt from the masked data.

	PERSON_ID	EMP_NO	FIRST_NAME	LAST_NAME	FULL_NAME	DATA
1	100	1000777	Robert	Smith	Robert Smith	A67e
2	200	1000888	Sue	Jones	Sue Jones	B363
3	300	1000999	Thomas	Wells	Thomas Wells	C678

The issue here is that the operations which mask the `FIRST_NAME` and `LAST_NAME` columns are independent and can operate in parallel. However the operation which performs the synchronization must not begin until both the `FIRST_NAME` and `LAST_NAME` column in each row have been masked. If it synchronizes the data too soon, then the `FULL_NAME` column will be updated with one or more unmasked data values. It is very important that if parallelization operations are enabled that the software performing the masking has the capability to schedule its masking operations based on the completion status of other operations. It is also very important for the designer of the masking operations to be aware of this requirement and to properly configure the software for this purpose.

Data Masking Issues

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.

Table-To-Table Skips

One of the most common synchronization operations is Table-To-Table Synchronization. This technique is discussed in detail in the Data Masking Techniques section. The synchronization technique uses a set of join columns to determine which rows are related to each other and then the masked data is copied from the source table to the target table so that the rows are updated appropriately.

The problem with this approach is that if there are rows in the target table which have join keys which are not in the source table then those rows will not get updated as part of the synchronization process. As an example, consider the two tables below.

CUSTOMER		CUSTOMER_NOTES	
CustID,	CustName	CustID,	CustName
100	Smith	100	Smith
200	Jones	200	Jones
300	Miller	3001	Miller

After the masking operations complete on the `CUSTOMER` table `CustName` column these changes must be replicated to the denormalized `CustName` in the `CUSTOMER_NOTES` table. The problem is that there was a typo in the data entry process and the `CustID` of 300 is actually entered as 3001. If a simple Table-To-Table synchronization operation is performed the value of “Miller” in the `CUSTOMER_NOTES` table will not be updated and will still be present in its unmasked form. This is called a Table-To-Table Skip. The best way to avoid this issue is to mask all of the `CustName` column contents in the `CUSTOMER_NOTES` table with a simple value like “ABCD” and then perform the synchronization. If that is done, then no data will be left in its unmasked form.

Cleaning up the Data

Data Masking operations are designed (by necessity) to update the data in the target database with values that are benign. One issue to be aware of, particularly if Substitution techniques are used, is that the act of masking the data can remove special cases from the data. For example, if a dataset of pre-prepared last names does not contain any names with umlaut characters (ä, ë, ï, ü) and the existing data does contain them, then such characters will not be present after the masking operation is complete.

Sometimes this “cleaning effect” matters and sometimes it does not. It is important that the person designing the masking routines be aware of the possibility. There are a number of ways of countering this effect. An appropriate choice of dataset or the use of a Shuffling technique are two possibilities.

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. In a small enough organization an unmasked birth date could readily be attributable to a specific individual.

Whether this issue is important depends the intended use of the data and also on how the remainder of the information is masked. For example, perhaps it does not matter if a specific record is known to be associated with the CEO as long as the name, contact details and salary are masked.

Relevant Data

In test systems, most data will eventually appear on a front end screen and be visible to the end user 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 the end users of the system if random collections of meaningless text are used.

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. For example, no matter how elaborate an algorithm you develop to sanitize names in a table containing disciplinary letters you will always be vulnerable to an unknown nickname, or descriptive epithet being left unmasked.

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. This particular technique works because most test, training and dev teams have certain favourite records which they use for various purposes. For example it is usually not much trouble to find out which records a training team use when demonstrating a letter of commendation and ensure that a carefully hand sanitized version is available there. The remaining records can be random gibberish (or null) and it will not matter.

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.

Sparse Data

Not all columns in a table have data in all rows. For example, the `PREVIOUS_LAST_NAME` field in an `EMPLOYEE` table will probably be mostly empty. In this case, when masking the data, it is not appropriate to fill in every `PREVIOUS_LAST_NAME` – the majority must remain null. If data is added where none was previously present, then the size of the table massively expands, there can be serious issues regarding storage and row chaining and, perhaps most importantly, the masked data does not actually reflect the original values.

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 the association of actually having had a previous surname 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.

Sequence Generation.

Occasionally data masking operations require sequences of values to be generated. For example, if a `CUSTOMER_ID` value is being masked, it might be desirable to have the new values be generated in a sequence such as ABC10001, ABC1002, ABC1003 etc. The ability of the masking routines to generate sequences of data is sometimes quite useful.

Consistent Masking

Often the end users of the masked database require the identical values to be substituted in the same fields each time a newly masked database is created. For example, if customer 10001 has a masked name of "John Smith" the first time they see it they may well require that record to have the name "John Smith" every time a newly masked database is given to them. Training teams, who generally have a standard set of actions to perform, particularly seem to need this. It pays to be aware of this requirement and, if required, ensure the data masking routines which are implemented can support this.

Aggregate 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? An awareness of the aggregate values of collections of information present in the database is a consideration when deciding on which masking operations to perform on a database.

Meta Information

It is not only the data in the database tables which may need to be rendered anonymous. What if the structure of the database itself is sensitive? For example, if there is a table in the database entitled `PROJECT_XYZ_NOTES` then this is a sure indicator that a project named XYZ exists and this visibility may not be desirable. The presence of such "information about information" is a judgement which has to be made when designing a set of masking operations.

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 and time – the deeper operations the harder and longer it takes to maintain synchronization. This issue is discussed in detail in the *What to Mask* section of this paper.

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 *Number and Date Variance* techniques previously discussed are the tool to use.

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* operations 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. Note that this issue is not quite the same thing as discussed in the *Cleaning Up The Data* issue. This particular case is much more about special data relationships which might need to be preserved after masking rather than the specific content of the data items.

User Defined 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. A thorough analysis of the user defined field contents will always be 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 Table-To-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 – it may be necessary to build indexes on-the-fly specifically for the purposes of synchronizing the masked data. Alternatively, speed improvements can often be achieved by dropping certain indexes, triggers and foreign keys for the duration of the masking operations. The data masking routines should be able to perform these operations. Also, and it is just as important, once the masking operations are complete the data anonymization routines must be able to remove any temporary indexes which were built and then rebuild any indexes, foreign keys and triggers that were dropped.

Repeatability

It is probable that any sanitization operations performed 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. It is usually reasonable to expect to take a bit of time developing a set of masking operations. However, once they are built it is very desirable to be able to initiate those routines with little or no operator intervention. Costs quickly add up if the masking process causes a team of people to have to work over a weekend or delay the availability of test or development databases.

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. As was discussed in this paper, there are a variety of techniques available, and the process of data masking has an even larger number of traps waiting for the unwary. It is important when sanitizing data in test systems that a thorough analysis of the requirements be performed. Very often there are a considerable number of trade-offs which must be made. Some questions you may wish to consider as part of your analysis process are:

- Who will be the end users of the masked data? Is it possible that different levels of masking will be needed for different groups of end users?
- What sort of depth of masking (granularity) of the anonymous data will be required in the resulting databases? For example, if all of the other Personally Identifying Information (PII) is removed is it actually necessary to render the CUSTOMER_ID number field anonymous?
- How large is the data to be masked? If there are many rows of data it might be necessary to scale back the amount of masking and synchronization operations just so that it is possible to perform the masking operations in a reasonable amount of time.
- Is there a time window during which the masking operations must be completed? An example of this would be that the masking operations have to fit into the time window available for a clone and backup cycle.
- Which data should be masked and in which tables and columns are these data items located?
- What masking techniques will be used on the various datatypes?
- Are the data items to be masked involved in any relationships with other data items? In particular look for foreign keys or logical relationships to other tables. However, also consider the possibility of Row-Internal or Table-Internal Synchronization requirements. Are there Cross Datatype, Cross Schema, Cross Database or Cross Platform synchronization issues involved in the masking operation?
- Are there any of the data masking issues (discussed above) applicable to the anonymization operations? In particular look for the Isolated Case Phenomena, the Where Clause Skip and the Table-To-Table Skip data masking traps.

Good luck with your data masking! We hope this whitepaper will provide you with some useful topics to consider. If you have any questions about the contents of this paper or data masking in general please do get in touch with us. We are happy to provide advice and assistance.