Overview

Function

There are many situations where a database needs to be 'exported' though the data must be sanitized to remove personal and business information. DataBender is a tool to do this securely and flexibly. For example the initial stimulus was to be able to show potential clients database systems that I'd written for others. Almost any organisation that uses live data will want a simple and secure method to create dummy data. Any such system will need to be keyed-in to compliance rules regarding personal and confidential privacy.

Uses

Allowing training, demonstration and system development on a fully functional and representative database without exposing confidential details or business statistics.

Providing a secure, reliable and verifiable data export mechanism.

Creating pseudo datasets for training, demonstration and system development and providing simulated export data.

Key features...

Exporting is a *controlled* process from start to finish.
Business needs are documented and drive the technical implementation.
Compliance with good practice is demonstrable.
Efficient and accurate physical exports according to defined procedures.

...Deliver

Leak-blocking awareness... ...and no excuse not to implement it.
Auditable procedures

Necessity

There are many cases where large amounts of private information have been exposed by large organisations through incompetence, indifference and slippery-nature of database administration. This affects smaller organisations as well who can hold a

surprisingly large amount of confidential and compromising information. Anybody who looks after almost any data is sooner or later going to be confronted by the need to allow access beside the officially secure everyday access methods. For example a travelling salesperson may need some personal details on their laptop but only relevant records and only partial data. Or an external auditor may need records which need to be anonymous, representative, and yet traceable should specific situations need to be followed up. Someone might want to analyse types of transactions statistically, so all confidential data needs obscuring first. Expansion into a new country may need 'a safe clone' of the database to get started, firstly for demonstration, then customising, then training.

In all of these cases a database administrator might be tempted to allow export of data without any filtering, simply because they don't have the tools tied-in to their compliance needs. Compliance officers might be tempted to follow the path of least resistance and trust the third parties rather than make a difficult database strip-down a requirement. We've seen this happen at HMRC in an unbelievably slap-dash fashion even when the auditors requested personal details at least obfuscated.

With the advent of DataBender there are no excuses for letting any data out of the database without implementing a security policy for that purpose. It is simple to configure, keyed to security policies, flexible and tunable and verifiable.

Example

Suppose that I am selling medicines to private individuals. I want to create a dummy system that I can use for staff training. Also from time to time I review my product range in the light of turnover and customer profile to inform my business strategy. There may be thirdly and fourthly and morely from auditors and regulators, but that's enough for now.

We can now start the *business* analysis

1. I have some data in a database. I know roughly the structure and significance of the record contents.

   CUSTOMER : Name, address, gender, age, our-ID (etc)
   TRANSACTION : Product-ID, Value, Qty, Staff-ID, Encrypted cc, Date (etc)
   PRODUCT : Product-ID,Supplier, Name, Drug class, Cost price, Retail price (etc)
   STAFF : Staff-ID, Name, Qualifications, Role, Password-Hash (etc)

2. I have some purposes for which the exports are being made
   Purpose 1: Training (and system development)
   Purpose 2 : Sales profile analysis

Now where am I going to *document my security policies*? Why not let DataBender help me by providing data-related, and purpose-related hooks. By the way, notice that some of the security leakage I want to avoid is record-based (eg How old is Mr Jim Smith and what medications does he take) and some is table-based (eg What is my monthly turnover and profit.)

DataBender can discover the structure of the database (ie. database schema) and use that to start documenting what we're going to do with the data. The latter will vary from purpose to purpose. For example for training I might specify CUSTOMER records to be limited to 50 and have all real name and address data replaced with fake data and shuffle the our-ID values (to prevent cross reference lookup from training to real system). For the sales profile analysis, which I take home on my laptop to play with, the same considerations apply without the table size limit. The training purpose will need completely fake STAFF but for my sales analysis I'll need to know actual details so I can see who sold how much. But I won't need any payroll or home phone numbers so should be sure to wipe or fake these before allowing the data out of the locked server room onto my loose laptop. We will continue with this example later.

Implementing bends

Even though DataBender hasn't actually hacked any data yet it should be obvious that we are already serving an important regulatory role of specifying the circumstances under which data may be 'allowed out'. Everything is in one place with variations and details being covered. The next step is to implement some actual 'bending'.

Every 'bend' can be regarded as a module that modifies either a table or records (possibly related). Simple table bends might be reducing the number of records to a random or 'representative' sample, or creating dummy data from pools of options. Of course, given the way tables are related we can expect these objectives are anything but simple when we want to tie up the loose ends. That's what a computer is for; we may have to give it heuristics but otherwise it knows where relations lead. Simple field-based bends may be used to obscure, remove or delete data by say replacing names and addresses with fakes. Again practice is more complex than theory where we may need to replace many cross-reference links together, or to keep the same data values but shuffle it amongst the records or reshape some statistics whilst retaining a degree of verisimilitude. We might want to add noise to make external cross references difficult while not damaging the purpose of the exercise. For example we may jitter dates of birth by plus and minus a couple of months and shuffle all the post codes so we have the same actual number of records per postcode but they aren't the real transactions.

Of course this sort of hacking is an art of obscuring details without jeopardising the value of whatever analysis exercise we're engaged in. One way of estimating the perturbation of the end result is to produce a number of sample data sets on which the analysis can be performed: If results concur then it looks like a safe method.

Example part two

Let's look at how we might bend the data for the purpose of creating a parallel training and development database that is realistic but 'safe'. We have already identified the fields to be protected now we have to implement the alterations.

Random replacement bending

For the CUSTOMER name and address we want real names replaced by false ones. To do this we'll need templates and pools of data to select from. (I call these sources of

inserted or replaced data 'cartridges'.) For our purpose the cartridge might be a simple text file with four fields which we'll select from randomly according to a template:

Cartridge data	**Templates**
John/Smith/1 High St./Wimbledon Mary/Jones/2 Laburnum villas/ Colchester Tim/Brown/123 Richmond Rd./ Sevenoaks (and so on)	CUSTOMER.Name = $1 $2 CUSTOMER.Address = $3,$4

Giving records like "Tim Smith","2 Laburnum villas, Sevenoaks"

The cartridge data doesn't have to be arbitrary text, we might instead be selecting from another table (ie. database data based) or enumeration (ie. database schema-based). Suppose for example that we were substituting the actual product purchased into the TRANSACTION records then we would use the PRODUCT.ID field as our source.

In the example we decided that we shouldn't use real our-IDs as somebody might relate id number 456 in the dummy data to the real 456 after they've done their training. Now we want to do two things: Firstly substitute random (and different) numbers into the PERSON.our-ID field and secondly to distribute that replacement throughout the related tables making sure that the data such as transactions still hangs together even if the foreign key relation value has changed.

Shuffling record bending

We may need to retain actual data values, possibly a group of associated fields, while detaching them from their original record. In the example I might be very sensitive about how much trade I do with specific suppliers and definitely don't want any programmer doing a one line SELECT to discover this. To get round this I could do two things with the product table: Substitute the product name and substitute the supplier. This is hard work because I'd need to think up a new name for each product. Alternatively I could shuffle the product names - put all the names into a hat and reallocate them at random.

I could do the same with the supplier field as well. Reallocating supplier names means that • the system is still useable in everyday ways • statistical patterns (such as the turnover we do with my top three suppliers) are preserved • the real names might be guessable. Therefore shuffling may not be a very good way of achieving our security goals but it can be a good way to break actual data re-discovery. If I was giving data to a distribution company so they could quote for my delivery patterns then shuffling the post codes of customers (amongst other things) would retain the realism of the data.

Encoded and indexed substitution bending

A regulator may want to analyse my sales to spot signs of misuse or mis-selling of drugs. I'm not too keen on giving them access to every single trade but that's what they need to be able to spot dodgy staff or unusual patterns. So (amongst other things) I might replace the actual name of the drug with "Product 001", "Product 002" etc. and do the same for customer and staff names. If the regulator spots an abnormal pattern

and need the details they can come back and ask.

Of course it is possible to encrypt data values but there are business and technical difficulties which often make this a poor choice.

Jitter bending

As above, I'm not too happy about people knowing exactly what I pay for my supplies. Therefore I will alter the Cost Price and Retail Price in PRODUCT by 'something'. For a demonstration and development system I might say plus or minus 5 to 20%. Thus individual data records 'look about right' without being accurate.

Record weeding bending

Typically for a training or development system we don't want a large database. For one thing small tables are more easily tweaked. So we might reasonably set some targets for the number of records we want in key tables. An obvious way to do this is pick records at random until we've got enough. A more sophisticated method is to attempt to preserve or pervert some statistical characteristic. Suppose in my example database I decided to reduce the number of products to 20. This will have an immediate knock-on effect in weeding the associated redundant TRANSACTION records. DataBender can do this automatically. There is a secondary effect in weeding customers who have never brought these products. DataBender can make it simple to do this secondary weed.

Record expansion bending

For the purposes of generating test data it is often useful to use a sample of existing data as a basis for replication. Often test data is hand-crafted to meet specific needs, but sometimes it is convenient to create larger quantities of pseudo data. The developer of my example system might be required to demonstrate that complex procedures governing which staff can do what transaction are correctly implemented. Initial assurance might be by special test examples but after a while in service values, rules and parameters change and being able to use representative data rather than specially formulated data might be necessary.

Types of export

The top-level distinction between types of export is:

- All or large parts of the schema with some data
- One set of SELECT results

These are quire different in scope although bending will apply to both.

In addition DataBender has a built-in sub-setting facility so that multiple parallel exports can be created in one pass. We might use this to export monthly sales data to regional directors.

Packaging and distribution

The mundane physical aspects of export are a common source of data loss. Typically a junior clerk is given the job of copying data onto a CD then sending it to 'somebody'. DataBender deals with formatting, encryption, signing, credentials and key management, audit trail and managing sub-sets even to the extent of being able to automatically email sub-sets to designated addresses from within the original database. Furthermore this data manipulation, version control and logging are carried out in a designated 'secure' location with a clean-up afterwards to avoid leaking loose temporary files.

Why bother?

At this stage programmers are probably thinking that this sounds like a lot of hassle which could be achieved with a few lines of SQL and a bit of code to jiggle with random substitutions. Unfortunately that's not the *business solution* because we need a reliable system tied into the compliance requirements which is flexible enough to adapt and straightforward enough so that there are no excuses for forgetting, fumbling and even making things worse.

DataBender is a procedural firewall and as part of the security system is a high priority target which needs simple rules for operation in an environment that makes it difficult to hide unauthorised code. Part of this is specifying a standard operating procedure which means a data administrator has a clear set of steps to follow regardless of the purpose of the export. This means that operation and configuration can be independently assessed if required and also that new export requirements can be easily implemented with confidence.

Architecture

Hardware : Ideally DataBender would be run in a tightly controlled environment and use a separate database server. As many files as possible are in plain text. 'What goes where' is specified so that for example temporary files are always identified and deleted as appropriate.

Process : A description of the database is initially retrieved from the server. This is used as a framework for additional structural information, change recognition and security policies at all levels (database, purpose, table, purpose/table, field, purpose/field). This is followed by specification of bending methods and applying them. Applying methods could be done 'at the click of a button' when repetitive exporting is required. The exportable data is now available for export and all other working data can be erased. Additionally log reports and tests on the export data to confirm successful bending are available.

Data : Everything revolves around a database description. This contains policies, method parameters, result summaries and statistics.

Cascade and redundancy engine : One of the objectives is to ensure that the finished export is consistent. A simple example is deleting child records when a parent is deleted but in practice dealing with links is a very complex and sophisticated process. An easy to understand version of this is what do we do if being asked to remove the last/only child record? Should the parent be marked for deletion? The order and options involved in cascading are important and subtle.

The key elements are:

- Database describer that builds and compares differences the basic database description. This is 'pointed at' a database in order to extract schema information. When being re-run it looks for alterations. It provides a way for additional schema information to be set-up, for example it may not be possible to discover foreign keys.
- Policy documenter which uses the database description and one or more purposes to provide a structured set of security policies.
- Method specifier used to define the bending operations to be used.
- Extractor-Bender which may steal data from the main server. (Simply as a clone, but possibly bending the most sensitive data by omission before it reaches the DataBender environment.)
- Bender which does all remaining bending.
- Data analyser which performs analysis on selected fields 'before and after' to show the nature of changes and confirm bending has been carried out successfully.
- Utilities for creating cartridges, doing a data export, reporting and housekeeping.

All calls to bending methods are carried out using a defined namespace to enable plugin and derivative routines to be patched-in as required. This makes customised bending a matter of being able to write single purpose routines. (Using PHP in the prototype.)

Conclusion

There is a pressing business need to be able to sanitize database data and implement robust, efficient and appropriate physical exporting. To make sure this is done effectively it has to be driven by a managerial process using documented and quality assured methods. DataBender connects this process with the technicalities of doing a good job of sanitizing. A single tool serves all levels of involvement from supervisory management through to the junior who dispatches CDs. It deals with security policies, usage agreements, and making it easy for everybody involved to do the job correctly.

For more information contact Peter Fox : databender@peterfox.ukfsn.org