WHITE PAPER | JULY 2016

Moving Beyond Masking and Subsetting: Realizing the Value of Test Data Management

Huw Price
CA Technologies

Table of Contents

Section 1

Introduction

The concept of Test Data Management (TDM) might not be new, but it is often overlooked, under-valued and misunderstood, and the full value it can bring to the business unrealized. For many organizations and vendors, it is seen as an environmental issue only, and amounts to copying, masking, and possibly subsetting production data. This migrated data is then considered a "gold copy"; to be used across QA and development environments, and the goal of better TDM becomes the faster management of this copy, and meeting the need for new copies.

Synthetic data generation, if used at all, is used in isolation, and on a project-by-project basis. This is often underpinned by the assumption that data generation is good practice for an individual team or project, but is not viable across an entire enterprise. Modern enterprises usually have large, complex databases, with tens of millions of records stored in numerous formats and using disparate toolsets. The argument goes, then, that it is necessarily easier to copy existing production data, than to attempt to profile and model such complex data, as would be required to generate realistic synthetic data.

This assumption is, however, questionable, and organizations wishing to realize the benefits of better TDM should reassess enterprise-wide synthetic data generation, as well as how they store, manage and provision data. Synthetic data generation is not only more effective in terms of time, quality and money, but also often proves to be easier and more secure than fully masking production data - with the right technology, processes and structural team changes.

This paper contrasts the commonplace problems and potential benefits of TDM from an environmental perspective, considering the processes, technology and team structures often found at organizations.[1]

Section 2

The Drawbacks of a Purely Logistical Approach to TDM

For organizations that rely on production data for testing and development environments, masking and subsetting are mandatory practices. Masking is necessary in order to satisfy current data protection legislation, while subsetting production data can help reduce unfeasibly high infrastructure costs.

But, if TDM starts and ends with masking and subsetting, its real value for a business will not be realized. This can be shown on the basis of two assumptions: first, that the data collected by any modern organization will contain personally identifiable information which, by law, cannot leave production environments in a recognizable form; second, that the masked data is intended for testing and development purposes, and so must provide sufficiently for all test cases which must be executed, and for any new functionality.

Following these assumptions, it will first be shown why masking might in reality be neither the easiest, nor most effective way to provision secure data that is 'fit for purpose' for test and development environments. The team structures, technology and processes implemented by the typical organization will then be considered, arguing that the poor coverage of production data, in addition how it is typically managed, mean that the familiar pain points of project delay, budget overrun, and defects in production will persist, if masking and subsetting alone are relied on.

Masking is not the simple solution

First then, the view that securely copying and masking production data for test and development environments is the simpler option across a complex IT estate can be challenged. In reality, the effort of masking production data while maintaining referential integrity will often outweigh that of modeling the data, while effective masking requires that data is first profiled anyway. Further, certain complex aspects of the data are often left unmasked as a sort of compromise, so that masking might not be the most secure option in many instances, either.

When masking data for use in non-production environments, the inter-column relationships of the original data must be maintained, even when the sensitive content is masked. At a simple level, this means that referential integrity is maintained. At a more complex level, however, it means that complex column relationships are maintained. For example, if there is a “total” field, calculating the total of two or more other columns, these columns will have to remain aligned in the masked data. Most complex are temporal data relationships and causal relationships, for example a time-based total, which are extremely difficult to mask consistently.

What’s more, these relationships do not only need to be masked consistently within a database (intrasystem), but, for a typical large organization with multiple database types and complex, composite applications, they also need to be maintained across databases (inter-system). The more complicated the data is, the harder this becomes, while, at the same time, the data becomes easier to crack, as there is more information to be correlated.

Most often, data masking focuses primarily on content, and inter-system and intrasystem relationships, insofar as to maintain referential integrity. The complexity of maintaining content, column relationships and temporal relationships at the same time usually means that one is forsaken—often, for example, the temporal relationships between columns will be left untouched. Though this is not legally or technically sensitive content, this preserved information can be used to identify sensitive information, by correlating it with information external to the data.

Take the example of a set of masked transaction logs, where an attacker knows that a certain person made a transaction of a certain amount at a time. Though this information is not found in the sensitive content, which has been masked, temporal information might still be present in the masked database, since the consistent masking of content, numerical totals, and transaction times is extremely difficult. Once sensitive information has been identified in the first instance, it can be identified across databases and systems, as intersystem and intrasystem integrity has been retained. This, in effect, will deanonymize the data.

Even if the effort of profiling and masking the data in all its complexity had been undertaken, the data still cannot be considered properly secure, as information will still be preserved in its more complex aspects. For example, an attacker could identify the causal cascade effects of a transaction log by combining intrasystem, intersystem and causal information with temporal methods such as a snapshot comparison. It would then be possible to deduce how data is changed during an operation and also to deduce any causal cascade effects that may be present (e.g. any triggers firing on the database). Again, the masked data is only as strong as its weakest link: once this information has been identified, an attacker can work across inter-column relationships, deciphering commercially sensitive requirements, or, worse, personally identifiable information.

Considering the assumption that practically any organization's data will contain some personally identifiable information, masking presents neither a secure or efficient way to provision data to non-production environments. This is especially true given the growing demand placed on organizations to fully secure their data: $158 is the average cost per lost or stolen record.[2] The EU General Data Protection Regulation will make existing legislation more enforceable that forbids using personal information for any reason other than it was collected.

Considering further how the Symantec "State of Privacy Report 2015"[3] found that data security concerns determine where and how 88% of European consumers shop, and the risk of using masked data in non-production environments become unviable. Now that we have established the legal risks of migrating production data to test and development environments, we can consider the efficacy and efficiency of doing so.

People

Data Dependencies

It is uncommon for an organization to have a central team responsible for TDM, and instead individual teams are responsible for managing, finding, and making their own data. Teams therefore work in isolation from each other, in their own test or development environments, but often rely on the same data sources. This lack of centralization and collaboration leads to data dependency constraints, whereby if one team makes a change to a database, it affects every team. Frustration, delays and rework then follow, as tests fail for apparently no reason, with teams unable to identify whether it was caused by a defect in the code, or whether it was a data error. Teams often also lack the ability to version or parameterize test data, and the capability to roll back databases if another team cannibalizes them.

The SDLC is further viewed sequentially, as a series of linear phases, where one team finishes something, and passes it on to the next. As a consequence, development and testing teams too often find their projects stalled because of delays upstream. They are left waiting for data to become available, or for specifications and data feeds to be completed by other teams, hence, a significant amount of SDLC time can be spent waiting for data.

This lack of parallelism stands in stark contrast to the current drive at many organizations towards continuous delivery, where teams need data at every stage of the SDLC. It makes it impossible, for example, to efficiently perform end-of-life development on old versions of a system, while also using existing data for new versions. Instead, teams are then left with a new development environment, but without the data needed within it.

Technology

System Dependencies

In addition to data dependencies, hardware and system constraints are a familiar pain point for modern test and development teams. Applications have grown increasingly complex over the last two decades, with an ever-growing array of dependencies with other systems, both within and outside the organization. This often leads to barriers when wanting to test a system, as services might be either unstable or unavailable. Another concern voiced by test and development teams is a lack of full-size environments. Teams might find that another team has priority over the environment they wish to use, or might be unable to get the correct data because it's being used by another team. Though at first such constraints might not appear directly relevant to data, it will later be explained how better TDM infrastructure is a necessary step to their resolution.

Data Storage

Modern organizations store large amount of production data. Having copies of these databases, and running them on development machines, is both expensive and slow. High infrastructure costs are incurred by the storage of such data, including hardware, licenses and support costs. Further, servers are placed under increasing demands, as they are required to deliver high volumes of data to jobs simultaneously, while running open connections, sustaining open files, and handling data injections.[4]

Unless the way data is managed is re-assessed, such costs are unlikely to decrease. The amount of data collected and stored by an average business doubles each year,[5] and the advent of "big data" has meant that organizations now talk in terms of handling petabytes, not terabytes. As a consequence, data storage has become one of the fastest growing parts of IT budgets, with research suggesting that IT storage requirements are growing at a rate of 40 percent per year.[6] Organizations should therefore consider whether each copy of production data is necessary—given how some organizations tend to have multiple copies of a single database, the answer is likely to be "no."

Data mining and profiling

Without semi or fully automated technology, data discovery is one of the biggest challenges faced by teams wishing to deliver fully tested software on time, especially within an agile development context or in a continuous delivery framework. Testers can spend up to half of their time looking for data, forcing them to make an undesirable compromise between running every test required to prevent costly defects making it to production, and delivering software on time.

This is made worse by the inconsistent storage of data in uncontrolled spreadsheets (with little or no cross-referencing or indexing, for example), rather than in a centralized warehouse or repository. Further, databases are not usually properly documented, with organizations often lacking central dictionaries of test data attributes and related SQL queries. Data mining and allocation is therefore impaired, and teams cannot request data based on standardized templates or forms, or according to the specific criteria they need. As a consequence, they will often have to find a small set of suitable data by hand to match individual test cases and requirement—a time-consuming, error-prone process.

The lack of automated data profiling and mining tools also increases the risk of non-compliance. With data stored in uncontrolled spreadsheets, sensitive information might be found anywhere—for example in a notes column, when there was not an appropriate column field for the data, or if all relevant fields were already full. Without being able to search for specific fields, such information is less likely to be found, and so might make it into non-production environments. This goes against data protection regulation, which legislate that data can only be used for the reason it was collected. The total cost of an average data breach has risen by 29 percent since 2013, reaching $4 million,[7] and can severely damage a company's bottom line.

Process

Production data is not a "gold copy"

An inability to find fit for purpose data for test cases and requirements brings us to the greatest issue with using production data in non-production environments: it simply cannot cater to the majority of data requests that an organization is likely to receive, especially those coming from development environments. As described, test and development teams need data at every stage of development. A "gold copy" database, properly considered, must therefore contain a standard set of data with which to repeatedly test, and must contain the data needed to satisfy every possible test. Further, it should be production-like and up-to-date, including "bad data", as well as all previous data. Production data only satisfies two of these conditions: it is production-like, and might contain all previous data. It is not, then, a "gold copy."

Much production data is very similar, covering "business as usual" transactions, and is sanitized by its very nature to exclude data that will cause a system to collapse. It therefore does not contain bad data, and will leave new scenarios and negative paths untouched during testing. It is, however, these unexpected results, outliers and boundary conditions that normally cause a system to collapse, and it should be the goal of development and testing to test the edge cases, unhappy paths, and unexpected scenarios. Relying on sampling methods alone runs the risk of defects making it to release, where the relative cost of fixing them is 60–100 times more compared to detecting them earlier,[8] and can take 50 times longer to resolve.[9]

Further, data sampled from production is almost never going to be up-to-date, given environmental changes and the constantly changing business demands faced by IT teams. Because data is not stored independently of the environments, when an environment changes, it necessitates a system refresh, or a version update. This can cannibalize the data and scenarios that have been built from multiple production data sources, and useful sets of data can be lost. These then have to be tediously recreated by hand. Such system refreshes also typically tend to be immensely slow. For instance, we have worked with organizations where we have seen them take up to 9 months to complete this process.

Manually creating data by hand might provide a short-term fix, allowing for the immediate test cases at hand to be performed. However, because such data is made with specific requirements or test cases in mind, it will almost immediately become outdated. A good example of this is currency exchange rates or trading patterns. Here, data becomes outdated on a daily basis, meaning that manually created data cannot be re-used and so is usually "burned." New data then has to be laboriously created for each test, leading to mounting project delays, while testing rolls over to the next sprint, or is left to the next phase of the development lifecycle.

If effective masking across multiple databases requires that the data is first profiled, and yet still does not guarantee security or provide data of sufficient quality, it begs the question: why, once production data has been successfully profiled, would an organization not just synthetically generate the required data?

Section 3

The Ideal Alternative: People, Process and Technology

A better TDM policy involves taking a structured and, crucially, centralized approach to managing enterprise-wide data. Not only does this resolve many of the pain points detailed above, but it is in fact often a cheaper, more efficient, and even easier way to provision data to test and development environments—with the right technology.

Storing useful data as assets outside of environments, and pushing it into them on demand removes the environmental issues of TDM, which then becomes concerned with how the data is pushed back out. This, in turn, can be efficiently performed with a centralized approach to data storage, where data is modeled as reusable assets, which can be extracted as precise subsets, on demand. Rather than make do with masking and subsetting as necessary operations, synthetic date generation therefore presents a strategic objective, which, when incorporated within a broader TDM policy, allows for the delivery of fully tested software, on time and within budget.

Technology

Automated Data Profiling

It has already been argued that to effectively mask production data, it first has to be profiled, but even then the masked data will not be fully secure, and will not provide the data required for testing and development. Automated technology does exist, however, to reduce the effort required profiling data across a complex IT landscape. To profile the data, it is first necessary to "register" it, collecting as much metadata as possible. Such metadata includes table names, column names, and column types, for example, and exists even for non-relational database systems, in the form of copybooks for mainframe systems and mapping documents for fix-width or delimited flat files.

Once registration has been performed, mathematically based data discovery algorithms can be applied. With CA Test Data Manager (formerly Grid-Tools' Data Maker) for example, this is first done for individual schemas, identifying personally identifiable information (PII), and reverse engineering database relationships if necessary. When this has been performed, the systems can be joined together. In doing so, CA Test Data Manager uses cubed views to profile even the most complex relationships that exist in data, creating a multidimensional dataset where each dimension of the 'cube' represents an attribute of the data. This profiling allows an organization to understand exactly what data exists, where it is stored, as well as identifying any gaps in functional coverage.

Synthetic Data Generation

Having built an accurate picture of what data exists, and identified what additional data is needed for test environments, any missing data can then be automatically generated. As each real world can be thought of as another data point, data can be modeled and created to cover 100 percent of functional variations. This data includes future scenarios that have never occurred before, as well as "bad data," outliers and unexpected results. This allows for effective negative testing and for the development of a new system or subsystem. It provides a systematic way to test, so that unexpected results and scenarios that might not occur in the minds of testers do not cause a system to collapse, while defects are detected before they make it into production.

If there is not enough data to repeatedly test with, high volumes of data can also be created using automated technology. CA Test Data Manager provides an automated tool that works directly with RDBMs or ERP API layers, to generate data as quickly as processing power allows. Its bulking scripts can double the amount of data an organization has, as quickly as the infrastructure can handle it. In sum, such automated technology allows for the rapid creation of production-like data, with all the data needed to run tests, including negative testing, and enough data to run repeated tests.

Centralized Data Management

With the aforementioned technological improvements, an organization has already gone much of the way to establishing a “gold copy,” as defined in this paper (See Section, “Production Data is not a Gold Copy”). By further storing data, modeled as reusable objects, in a central Test Data Warehouse, they can also go about establishing the ability to rapidly identify specific subsets of data for test and development environments, on demand.

Data Cloning

Once data has been modeled as objects in a ‘Test Mart’ or Test Data Warehouse, and dictionaries of data assets and associated queries have been established, specific subsets of data can be identified, cloned, and pushed out to test and development environments.

The data cloning module from CA Test Data Manager, for example, pulls out small, coherent sets of test data from multiple, inter-related production and development systems, replacing the slow and expensive process of copying and moving large, complex databases. The ability to extract, copy, and provide only the data needed means that an organization no longer needs to maintain numerous full-sized copies of production databases.

Having a centralized data warehouse and being able to clone data can further remove data dependencies between teams, separating the provisioning and consumption of data. It means that data can be cloned and delivered to multiple teams in parallel, eliminating the delays spent waiting for ‘upstream’ data to become available, and preventing teams negatively impacting each other when they make a change to data.

Modeling and centrally storing data as malleable, reusable objects also allows for bugs and interesting scenarios to be easily reproduced. With data explicitly drawn out in bug reporting, flexible, rapid cloning technology allows complex and rare tests to be repeated, without the data getting used up. This is particularly valuable when conducting a data refresh, as it means that data does not need to me merged, and interesting data sets are not lost.

Hardware constraints

Combined with a virtualization toolkit, synthetic data generation can also help resolve hardware and system constraints. Message layers can be simulated, using the metadata of a system to accurately profile and generate realistic service messages (including Soap, REST, and MQ files, as well as Flat Files). Here, an automated data generation engine sits beneath a virtual machine, to populate realistic message responses, such as request/response pairs.

The virtualization of entire machines further means that multiple development environments can be created. This means that teams can work in environments, even when interdependent components are unavailable, preventing upstream delays, while costly legacy systems and hardware can also be virtualized for testing.

Process

Parallel development and reusability

In addition to environmental issues, the ability to discover and clone data on the basis of specific data attributes, also resolves the other central concern of TDM, which is how data is to be provisioned efficiently to individual testers or test teams. It allows data to be requested, shared, and reused in parallel, on demand.

Having access to a centralized web-based 'test data on demand' portal, like the one provided with CA Test Data Manager for example, allows testers and developers to request exactly the data they need for the task at hand. When specific criteria (i.e., test data attributes) are submitted, the portal sends a job to the batch engine, which will then either find the appropriate data from back-end systems, or will clone the data and push it out. This removes the need to manually search for data, or create it by hand, thereby significantly reducing the time taken to fulfill data requests.

The more standardized the questions on the form, the better, as it allows teams to better re-use each other's work. For example, if you have synthetic data that has been created before, you could parameterize the input and expose this through drop down lists in the portal, allowing everyone to request the data, even if the test case is different. In addition to test data, data creation frameworks, unit tests, virtual assets and automation scripts can be stored and used as building blocks for future work.

Version Control

Powerful version control allows for the parallelism needed to continually develop new systems, on time and within budget. For instance, CA Test Data Manager's test data warehouse allows a team to copy data from a repository, which in effect 'inherits' it, along with pointers back to previous versions. This accommodates the evolution of data across multiple versions, as it locks data for a particular team, while allowing data to be easily rolled back or forward and reconciled with different versions. When a change is made in one place, it "ripples" up and down versions, while the original remains intact. Say a team needs to add a new column to an entire database: With CA Test Data Manager, if they have hard-coded parents, they can find all linked children and set default values, or can generate data using sequences or standard functions.

People

Finally, structural team changes can often supplement these technological and procedural improvements, and can further help resolve the discussed environmental pain points. The centralization of TDM under a dedicated team correlate to having central data storage, management and provisioning resource that can more efficiently serve the need of the enterprise. This team might own the provisioning of data, as well as the management of data itself, and could further be responsible for building new data and profiling data if necessary.

Not only does this help avoid data dependency constraints between teams, it also means that data requests and bug reporting can be brought under one remit. Quality gates can therefore be enforced, while data ownership can be centralized under the IT security team. The dynamic form building offered by CA Test Data Manager's Test Data on Demand portal supports this, as it goes beyond role-based access rights, only provisioning sensitive data to the authorized staff who request it.

Section 4

References

1 The drawbacks of actually using production data in non-production environments have been addressed elsewhere. See, for example, Huw Price, "Reduce Time to Market with Test Data Management" and "How better Test Data Management is the only way to drive Continuous Delivery"

2 Ponemon Institute, "2016 Cost of a Data Breach Study: Global Analysis," June 2016, https://www.ibm.com/marketing/iwm/dre/signup?source=mrs-form-1995&S_PKG=ov49542

3 Symantec, "State of Privacy Report 2015," https://www.symantec.com/content/en/us/about/presskits/b-state-of-privacy-report-2015.pdf

4 Jacek Becla and Daniel L. Wang, "Lessons Learned from managing a Petabyte," P. 4. Retrived on 19/02/2015, from http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/

5 Ibid

6 Jason Buffington, "'Clear Skies Ahead' for Hybrid Cloud and Tape Solutions," Enterprise Strategy Group, March 2015, http://strongboxdata.com/pdf/sb/CRDS_StrongBox_ESG_Solution_Showcase.pdf

7 Ponemon Institute, "2016 Cost of a Data Breach Study: Global Analysis," June 2016, https://www.ibm.com/marketing/iwm/dre/signup?source=mrs-form-1995&S_PKG=ov49542

8 P. Mohan, A. Udaya Shankar, K. JayaSriDevi, "Quality Flaws: Issues and Challenges in Software Development," http://www.iiste.org/Journals/index.php/CEIS/article/viewFile/3533/3581

9 Software Testing Class, "Why Testing Should Start Early in Software Development Lifecycle," http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/

Section 5

The CA Technologies Advantage

technologies

Section 6

About the Author

With a career spanning nearly 30 years, Huw Price has been the lead technical architect for several U.S. and European software companies, and has provided high-level architectural design support to multinational banks, major utility vendors and health care providers. Voted “IT Director of the Year 2010” by QA Guild, Huw has spent years specializing in test automation tools and has launched numerous innovative products which have re-cast the testing model used in the software industry. He currently speaks at well-known events internationally and his work has been published in numerous magazines such as Professional Tester, CIO Magazine and other technical publications.

Huw’s latest venture, Grid-Tools, was acquired by CA Technologies in June 2015. For nearly a decade, it had already been redefining how large organizations approach their testing strategy. With Huw’s visionary approach and leadership, the company has introduced a strong data-centric approach to testing, launching new concepts conceived by Huw such as “Data Objects,” “Data Inheritance” and “A Central Test Data Warehouse.”

CS200-211197_0716