A Best Practice Guide
Data Strategies for Application Testing

Original Software

Introduction

This best practice guide explores strategies and techniques which have been used to improve both test efficiency and software quality for applications based on IBM i Databases, utilising methods to create, manage and validate test databases.

The profile and significance of Software Quality has increased through commercial and compliance pressures driving the need for IT teams to simultaneously reduce risk and delivery timescales. A basic building block of a coherent strategy for Software Quality is appropriate and accurate test data.

This guide explores the key principals and techniques as they relate to a test environment for targeted data extraction, data de-sensitisation, use of pair-wise optimised input scenarios, data synchronisation with test scenarios, as well as simultaneous user interface and database validation.

Overview

Test data environments

Full size data is usually unwieldy, slows testing and makes it more difficult. The ideal test database combines essential background data and reduced, targeted, transactional data representing, as much as possible, a good cross-section of the live environment. The data extraction component of TestBench is designed expressly to address this need. An appropriately constructed environment will save time in testing and improve the ability to detect errors, raising quality.

Data confidentiality

Apart from legal requirements, common sense and good practice says that exposing traceable production data to test disciplines is risky. Yet, production data as a source of background data is valuable and desirable. Using a data

obfuscation technique such as TestBench's provides the best of both worlds, real data that cannot be traced and does not represent a security risk.

Synchronised, protected data

"TestBench's "Scrambling" provides the best of both worlds, real data that cannot be traced and does not represent a security risk."

One of the biggest time-wasting factors in the process of performing testing, is dealing with corrupted or unsuitable data. The ability to synchronise the state of the database with the initial requirements of a test case has saved weeks in projects where it has been employed. We have put this capability in the hands of users and testers with an "Environment Protection" module.

Database validation and verification

In combination with testing the user interface, there is a stage in the software development lifecycle that the integrity of the database needs validation as well. This is often a technical challenge requiring both detailed application and tool knowledge. Typically, not only do you need to know SQL for example, but also in which tables to look, and what the data means. Testers should be able to see detailed effects in the database made as a result of a test without having to go searching for it.

Extending this concept with pre-defined Data Rules ensures automatically that all database events comply with business or test-based rules.

What Data?

"The ability to synchronise the state of the database with the initial requirements of a test case has saved weeks in projects where is has been employed."

As part of a test strategy, whether in support of functional, regression, system, or unit testing, decisions are required to provide both base data and input scenarios. Input scenarios are dependent on the base data but in practice when considering what data to use to populate a test environment, the base data should be dependent on the input scenarios.

So, the first thing is to identify what are the scenarios that need to be tested. In the case of functional testing or new developments, this has hopefully been identified early on in the requirements and functional specifications. Whilst you may consider regression testing as separate phase, it is common to perform targeted regression testing around impacted functionality to ensure that existing capabilities are not incorrectly affected. This in turn leads to the need to

understand the data requirements which are able to support a test of all the specified capabilities (new and pre-existing) of the function being tested.

In addition, it may be necessary to consider data volumes and quality. Large volumes of test data, whilst necessary for stress or load testing, inhibit functional or regression testing. Where possible, a sub-set is more effective and is easier to validate. However, it is not simply a case of taking ten percent of records as referential integrity has to be maintained.

What Options?

Your test environment must accurately represent the live environment. Having a test environment that's quite different from the real environment is not effective. Broadly there are four options for building a test environment.

1. Take a full copy of production data if that exists
2. Take a subset of production data
3. Start with empty files and set up the required data
4. Programmatically create data

Or combinations of the above. There are advantages and disadvantages of each of the above such as: -

1 – Full copy of production data	Too large, taking up space and extending processing time Time consuming to create the test environment May contain sensitive or personal data Cost – often a separate server to host data for testing
2 – Subset of production data	Difficult to build a subset which maintains referential integrity Needs rules and thought as to what to extract/include May contain sensitive or personal data
3 – Empty files and populate	Time consuming and labour intensive Needs rules and thought as to what to include Does not contain real data found in production
4 – Programmatically create data	Unlikely to look and feel like real data Difficult to create with referential integrity Does not contain real data found in production

For these and no doubt other reasons, people will often shy away from the problem as it is difficult to solve well and efficiently. Sadly, this can mean that the subsequent testing activities are devalued and hindered by the lack of ideal test data which is a fundamental building block of a good test strategy.

It also highlights the key fact that if you have invested in creating good test data, it is something of significant value and should be treated as a corporate asset.

If you've gone to all the effort of creating a full test environment, why not hang onto it for ongoing testing? This could be a test environment that is used to test version updates and core functionality changes or to provide a training environment. Just be aware that there may be licensing considerations with a test environment for continued use.

"The subsequent testing activities are devalued and hindered by the lack of ideal test data which is a fundamental building block of a good test strategy."

A combined strategy for creating ideal data environment

Taking into account the 4 basic methods on page 3, for many situations an optimal strategy is probably based on the following outline.

1. Breakdown the application data into different categories
    a. Base data, that is required in their entirety.
    b. Old or redundant records can be discarded.
    c. System logs, audit data, does not require any data.
    d. Transactional data needs to be reduced.
2. Use a data extraction solution or copy tables in the 1.a. and 1.c. categories above.
3. Using a data extraction solution to
    a. Target 1.b. above to get the current master data.
    b. Target 1.d. above to populate a percentage of data. A typical result might contain 20% and 1% of original.
    c. Target 1.d. above again, to obtain a mixed sample of records across key values; for example, 10 records of every available combination of product group, tax code, currency, order type and delivery method.
4. Supplement this data with other situations needed

a. Via data entry.
b. Programmatically.
c. By adapting existing data to new scenarios.

5. Apply scrambling to sensitive data to de-identify it.
6. Back it up!
7. Employ data management to enable recovery and reversion to prior situations to negate data corruptions and accidents in the testing process.

The Challenges

One of the practical challenges with using a data extraction solution is in the creation of an entity relationship model which adequately defines the data hierarchy so that extraction criteria can be defined in an accessible way. Information should be taken from the tables to build this hierarchical model automatically, thus speeding up this phase and making the resulting Data Cases business friendly. Using this technology enables a suite of appropriate Data Cases to be built, maintained and adapted to service the data extraction requirements for creation of a coherent test environment.

Data Confidentiality

Legislation covering the use of data, especially personal data exists in many industries and is increasing. In the US, the Privacy Act of 1974 covers confidentiality of personal data, and more specifically HIPAA addresses the Health Insurance industry, the Gramm-Leach-Billey Act and FSA (UK) addresses Financial Services. In various ways these prohibit the use of data outside of secured production systems which can be traced back to a real individual. The General Data Protection Regulation is recognized as law across the EU with global implications as to the protection, accuracy and legitimate use of personal data. So typically, we are in a situation where the company is obliged to ensure the accuracy of data which effectively excludes its use in most test situations. Auditors are often identifying the use of live data in testing as a high-risk exposure and many organisations are left with audit actions to address this concern. Various additional laws have been drafted and the topic is certainly set for greater constraint and legislation.

Whether legislation is a concern or not, it has commonly become unacceptable to expose traceable production data to persons who do not need access to it with increased risk of loss, misappropriation and accidental release. In addition

using real data in a situation where it may be altered and hence become inaccurate may lead to a situation of an individual being identified with false associated information.

Thus, if production data is being used it needs to be amended to remove the key information that makes it identifiable. Typically, this will involve data items such as name, address, telephone numbers, SSNs, credit card numbers and so forth. It may be applicable to neutralise this with programmatically generate values such AAAA00123, AAAA00124 etc or to use a mixing algorithm to scramble data and hence remove traceability.

In this case, by taking data from a number of records and mixing first name, surname, first line of address, second line, town, phone number etc. it is possible to generate a fictitious person at a fictitious address, but with data that is perfectly acceptable for testing. This approach we refer to as scrambling and for added security techniques such as scrambling parts of fields (such as credit cards or SSNs) provide greater protection.

Using scrambling means that realistic data is available which is more comfortable to test with, especially for users, yet it is sufficiently anonymous to provide protection from exposure or legal pressure and programming is not required.

Data for Input Scenarios and Data Driven Testing

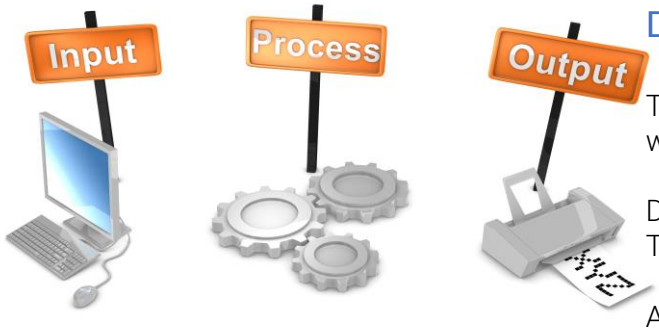
In the planning stage for testing, it is necessary to consider what data is to be used to drive the test scenarios. It is relatively easy to determine a suitable list of values for each data item which may not need to cover all possible values depending on the nature of the data and its significance in the process.

It is harder to determine how to combine values from different columns into a viable test set. For example, if the key items in a process are: Order Type; Currency; Method of Dispatch; Tax Code and Customer type and if each of these columns has just 6 possible values, the data set required to test this completely would consist of 6^5 (six to the power of 5 = 7,776) tests. Despite this representing a relatively small data set, the extrapolated tests are probably already too numerous to be

carried out in full. With more realistic examples involving additional columns and rows, the exhaustive data set quickly grows into the hundreds of thousands or millions.

Pair-wise or all pairs is a well-documented strategy which has been used extensively by NASA for creating good coverage of scenarios from situations where exhaustive testing is not possible. It is based on the principle that most errors will be found from testing:

a. values on their own
b. every pair of values combined.



Data Synchronising – Protecting the Assets

The core elements of the process of testing are synonymous with the definition of data processing itself.

Data processing	*Input – Process – Output*
Testing	*Input – Process – Result*

A test case should be able to define these three elements including the predicted result. Hence a prerequisite for this definition is the data involved. Appling the same data to an unchanged process (logic) will produce consistently the same result which is easily checked.

Consider a different but common situation encountered in testing. The data has changed, the processing may have changed and the result is different. The challenge for the tester is to determine if the changed result is compatible with the changed data or is a result of the changed process, or both. This inevitably makes automation of such a test very challenging and requires the same logic found in the application to be reflected in the test.

However, if it has been possible to retain the same data, one of the variables is eliminated and immediately any difference can be tracked back to the processing. It also becomes a test which is much simpler to automate and to check automatically – is the result the same as before (the baseline)? If not, is the difference accounted for by the specification of the change, in other words is it a change we are looking for or is there a problem?
If it is possible to apply this philosophy, especially to regression testing, the rewards of time saving are immense. It

is possible to re-invest the time into extending and improving the regression tests and hence reduce risk by finding more errors.

Data backups are an obvious way of addressing this and may be an appropriate technique. However, in large environments they can be time consuming, space hungry and require specialised operational resources. Restoring can be even more painful in such situations.

Original Software has developed a unique technique which negates the need for backups, takes moments to perform and can be carried out by the data users themselves, on demand. It enables the test team to synchronise the state of the database with the required conditions for the test scenario being performed. This capability has been proven to save weeks in typical test cycles.

The technology uses the MS SQL Server and DB2 infrastructure to capture the state of the database at any given time as a “Checkpoint”. This can be performed on demand by the user or integrated in test processes and steps. Once a check point is set, testing continues and should there be an error, data corruption or discovery of an omission, the database can be very quickly rolled-back to the latest, or any prior checkpoint and the problem resolved. This approach removes the need to keep on working with imperfect data, add additional data or resort to a time consuming restore (assuming there is one at a suitable point in time).

Visibility and Verification of the Database

The most natural and obvious place to focus testing is around the user interface. It is a window into the database and the business processes being performed. In addition, the database itself warrants attention although the degree of attention will decrease as later phases of testing, such as User Acceptance Testing, are carried out. However, early on it too requires significant inspection in order to validate its integrity as well as the performance of the functions affecting it. With some systems, such as those based on a straight-through transaction processing model, the database maybe one of the few places where testing is possible.

“The ability to combine simultaneous testing of the user interface and the underlying effect on the database is a powerful combination.”

The ability to combine simultaneous testing of the user interface and the underlying effect on the database is a

powerful combination. Technology should show the tester, in a clear and easy to understand format, exactly what has happened in the application tables whilst the test scenario was applied (itself via an automation solution) to the user interface. In other words, both the user interface and the impact on the database can be simultaneously verified.

SQL can be a partial substitute for this approach but has a number of disadvantages which reduce its effectiveness and use. These are:-

- Knowledge of SQL itself is required, which limits the audience to technically advanced users.
- Knowledge of the database is required to know how to retrieve the required data and from which tables.
- Knowledge of the database structure is required to retrieve related data in other tables.
- Knowledge of the applications impact on the database is required so as to know which tables are affected.

A more powerful, more widely useful and much easier to use Capability is delivered in TestBench which very simply shows the rows in every table that has been affected by the test process, with before and after images of each column. This means that the user does not have to find the data or know which tables to look in and that they can see exactly what has changed on each occasion in chronological sequence. For example this could start with an insert, followed later by an update to some columns in the same record and further updates to other columns later. Another significant advantage is the fact that this technology does not depend on knowing which files to look in, it simply presents all the effects in all the tables altered by the related test process in a clear and easily understood format. This is typically deployed with user interface automation utilising TestDrive.

This is significant in testing as residual errors often lie undetected because that area was just not checked, perhaps because there was no knowledge that it needed looking into. It is analogous to the rather obvious observation that whenever you are searching for something you find it in “the last place you looked”. Of course, why would you keep looking having found it, but you had to know where to look in the first place.

Proactive Validation

Having exposed automatically the effects in the database it enables validation of this data to be taken to a further level using *Data Rules*. This provides the tester and the developer with the capability to define rules either generically for the table or a specific situation expected in a table as a result of the test being performed.

Rules, once set up at the table level can be inherited by other test cases so that a library of Data Rules is accumulated and increasingly checks key aspects of the tables' properties during every test. Using this technique, it is possible for each write operation to the managed test environment to be checked firstly to see if there are any applicable rules and secondly if any of the data values in the row break conditions specified in the Date Rule. This means that the tester is proactively notified of database events which occurred in contradiction to the rules, without having to go and look or without having to run any SQL.

Total Testing

"Without TestBench we could not have installed the software in time. TestBench enabled a business-critical system to be live, on time, with full quality assurance."

This innovative approach to software quality is a key component of what Original Software refers to as "Total Testing" and has been implemented in a number of major international businesses. In particular, the creation of targeted test environments has been used to increase coverage and understanding of business related test issues whilst simultaneously improving time to perform tests and increasing error detection rates. The capability to synchronise the condition of the database with the matching test step and scenario using a controlled checkpoint and roll-back strategy has been proven to dramatically reduce testing timescales. The effect is apparent even from initial implementation and while more advanced code-free automation strategies are developed which in turn provide on-going time savings and increased coverage by re-use of data and matching scenarios. Finally, the ability for the tester to simultaneously and clearly validate all aspects of the user interface and the associated database updates provides a repeatable way to test in-depth and deliver robust applications in a shorter timescale.
This technology enables a much wider view of what is

important in testing (beyond the user interface) and provides a strategy for real time saving and greater quality.

About Original Software

With a world class record of innovation, Original Software offers a solution focused completely on the goal of effective application delivery through quality management. By embracing the full spectrum of application quality management across a wide range of applications and environments, the company partners with customers and helps make quality and efficiency a business imperative. Solutions include a quality management platform, manual testing, full test automation and test data management, all delivered with the control of business risk, cost, time and resources in mind.

More than 400 organizations operating in over 30 countries use Original Software solutions. Current users range from major multi-nationals to small software development shops, encompassing a wide range of industries, sectors and sizes. We are proud of our partnerships with the likes of Allianz, Bimbo Bakeries, Costco, CertainTeed, Delta Dental of WI, Euronet. IAT Insurance, O'Reilly Autoparts, Cayman National Bank, Topcon, and DSC Logistics.