Test Smarter

DELIVERING APPLICATIONS THAT WORK

WHITE

PAPER

June 2001

princeton softech

Contents

Why Test Smarter?

Application testing for accuracy, reliability and quality has never been more important. Why? Because the applications that once merely supported your business are today a critical element in operating your business. Companies depend on sophisticated applications for customer relationship management, electronic commerce, financial reporting and more. In turn, these applications rely on relational databases to store and manage the underlying enterprise data.

Now, more than ever, companies face new challenges when designing effective and efficient testing strategies for relational database applications. Incomplete or inaccurate test data means inaccurate testing, which can lead to disaster. Yesterday, when an application failed, a company was simply inconvenienced. Today, that same company may find its name on the front page of the *Wall Street Journal.*

Testing Smarter offers a Competitive Advantage

Testing smarter means streamlining the application testing process to be more efficient, consistent, accurate and cost effective. So why test smarter? Because quality applications supported by development "best practices" provide a competitive advantage. Testing smarter can deliver dramatic results in the following areas:

- **Application Reliability** — Today's applications drive revenue and empower sophisticated marketplace initiatives. Customers and partners interact directly with these "customer-facing" applications, which have a huge strategic impact. How important is application reliability to your business?
- **Time to Market** — Shorter development cycles must be achieved, yet quality testing takes time, and this often means that deploying new releases to customers, suppliers and employees is delayed. How important would it be to your business if you could significantly reduce testing time without sacrificing quality, thereby improving time to market?

- **Cost of Quality** — The costs associated with resolving application defects in the production environment are as much as 10 to 100 times greater than if these defects are resolved in the development process. How would your business fare if application defects were discovered in production? Could your company withstand the cost of business disruption resulting in the potential loss of customers, suppliers and revenue?

Testing smarter means delivering quality applications. However, some development organizations are not focused on planning and managing the testing process within the application life cycle. Testing is often considered a lower priority relative to development when, in fact, designing a comprehensive testing strategy can be as challenging as designing the application.

Many organizations are reluctant to make a significant investment in testing, allocating the money to other areas. However, without a commitment to quality assurance, the business consequences could be significant. Application testing is a requirement throughout the application life cycle. Any investment in an effective application testing strategy can increase return on investment (ROI) over the long term.

In today's market, quality is becoming a competitive differentiator. If companies want to be competitive, application testing must be a high priority.

Why is Application Testing a Challenge?

Typically, when a new application is developed or an existing application is modified, a test database is created, usually based on a clone of the production database. Special test cases may need to be added before testing can begin.

After test cases are executed, the results must be verified to ensure that the application program is working as designed. Based on the results of this verification, some problems may need to be corrected, and the test data would need to be refreshed before testing continues.

This process is repeated throughout the various testing phases (unit, integration, system, load, regression and acceptance testing) until the application is migrated into production.

The goal is to create a repeatable testing process that is automated as much as possible to improve application quality, reduce time to market and minimize costs.

Complexities of Testing with Relational Data

The fact that most applications rely on relational database technology introduces a major challenge for testing organizations. The application data model may contain dozens, hundreds or even thousands of tables, and just as many relationships, and data model complexity is not limited to large-scale systems. Even a database of less than a dozen tables can contain relationships that make navigating the data model difficult.

Without access to a generalized technology, the developers often have to code sophisticated extract programs. It is a challenge to navigate the many tables, rows and columns to create, manipulate and refresh the desired database subsets without first spending countless programming hours. And, because applications are always being enhanced, these extract programs also have to be maintained to create valid test data.

Afterwards, there is no clear way to know that the results are complete and referentially intact and no sure way to account for every possible relationship in the original database because they are often enforced by the application. Adding to the complexity is the challenge of handling heterogeneous database management systems, legacy data and differing data models.

Clearly, any comprehensive toolset must support these relational complexities and “remember” to account for them in every extract, compare or update operation.

Integrating Data from Multiple Database Management Systems

Many applications require integrating test data from different database management systems, such as Oracle, DB2 UDB, SQL Server, Sybase and Informix. For example, it might be necessary to create test data from an Oracle database running on a UNIX platform and include Oracle and DB2 UDB data from a database executing on a Windows NT platform. Adding to the complexity, all database management systems have different idiosyncrasies for handling data.

Heterogeneous systems are here to stay and evolving technology is a fact of life. Quality testing must include seamless capabilities to handle data from different database management systems operating on different platforms.

Integrating Legacy (Non-Relational) Data

While relational databases have inherent complexities, the task of testing an application that references both relational and non-relational data poses an even greater challenge. For example, an order entry application may require customer data from a DB2 Order Entry table and product data from a VSAM inventory file. Legacy data stored in VSAM files or IMS databases have different structures and cannot be integrated with relational data easily without writing special code.

In this type of environment, an effective testing strategy must include federated data access so data can be extracted and moved from legacy and relational data sources to create a true subset of related data regardless of the source.

Building a Test Database

Some of the more common approaches to building test databases include cloning the production database and writing custom extract programs. However, these methods are time consuming, labor intensive, error prone and often provide inconsistent results.

Typically, it is impractical to clone an entire production database comprising hundreds of highly interrelated tables just for testing purposes. First, there is a capacity issue. Second, there is a quality issue — when working with large test databases, developers may find it difficult to track and validate specific test cases. In addition, test databases shared by multiple users become corrupted, which hampers testing effectiveness.

Cloning an entire production database increases the time needed to run test cases because there is a larger volume of data. In addition, the production data may not be sufficient to support the specific test cases. It is much faster to test with smaller, realistic subsets that accurately reflect the production data without adding overhead to the testing process.

At the other end of the spectrum, some IT organizations rely on “toy” test databases that have been cobbled together by hand using custom coded extract programs. Usually these databases fail to reflect the actual complexity and variety in their production counterparts. The applications are tested and put into production only to break down when they encounter “live” processing conditions.

Creating, Editing and Masking Test Data

If test data is based on existing production data, it may be necessary to handle complex data models characterized by referential integrity (RI) enforced by the database and the application. Typically, the application enforced RI does not adhere strictly to the database rules, exploiting compatible (non-identical) data types, composite and partial columns, and data-driven relationships. Test data needs to be defined in a way that saves time and is repeatable.

Some techniques for identifying and creating test data include using selection criteria or random selection, data partitioning/grouping, or limiting the data by table or relationship. Without a generalized tool, any of these methods involve writing special programs.

When there is no existing test data, it must be synthesized, which involves more work. Sometimes a small set of data can be “multiplied” into more data. When creating multiple sets of test data from a single original set, it is necessary to modify primary key values to prevent creating duplicate rows. Propagating keys is a critical capability for synthesizing test data. When modifying the primary key value in the parent (or owning) table, it is necessary to propagate those changes to the child (or subordinate) tables to keep the relational sets intact.

Some application functions may require creating special test data to force error conditions. In addition, with the focus on privacy, the ability to transform or mask sensitive data is important. However, given the complexities of relational data, editing the data to create special test cases or masking sensitive data would be difficult without a capable generalized toolset.

Validating the Test Results

Validating test results and identifying changes after each test run are difficult at best. However, without a tool for comparing images of the test data before and after a test run, identifying all the differences is next to impossible. First, there are a variety of changes: inserts, deletes and updates spread across hundreds of tables. Second, there may be unexpected problems (for example, orphaned rows) as well as other anomalies that may go undetected.

An effective testing strategy must compare subsets of data to identify differences. With relational data this means more than comparing row to row. It means using data model intelligence to compare related sets of rows.

Maintaining a Consistent Test Environment

Maintaining a consistent test environment ensures quality. However, most cloned test databases are shared. This means that each time the test database is modified it diverges further from the baseline test data, resulting in a less than optimal test environment. It takes additional time and resources to refresh the data to ensure accurate results on subsequent test runs.

In addition, unless the extract process includes metadata, there would be no way to accommodate changes in the data model during the testing phases. Metadata is definitional data that provides information about the structure of the data managed within an application or environment. For example, metadata would document the structure of the database including the tables, columns, relationships, views, triggers and so on.

A well-designed testing strategy must ensure a consistent and accurate test environment. Working with predefined realistic subsets of data that can be refreshed easily improves testing and overall application quality.

What is Needed to Test Enterprise Applications?

IT organizations need access to technology that supports a smart testing strategy — one that creates realistic, referentially intact subsets of production data for accurate and efficient testing. They also need tools that can browse, edit, compare and manipulate these subsets of related data with speed and accuracy.

Selecting an Enterprise Testing Toolset

Any generally applicable enterprise testing toolset must address the following:

Comprehensive Testing Capabilities

As long as there are modifications and enhancements, testing is necessary at various phases throughout the application life cycle. The ability to deliver applications that are thoroughly tested is an ongoing commitment that requires comprehensive testing capabilities:

- A quality data migration tool should provide the capability to repeat testing processes consistently throughout the application lifecycle. This capability improves productivity because users can create and reuse realistic and manageable test data.
- Intelligent browsing and editing capabilities can reduce time spent on repetitive testing tasks.
- Comparison processing can identify differences in test results automatically and identify problems that would otherwise go undetected.

These tools should be readily available to all members participating in the development and testing process to ensure a fast, easy and resource-efficient test environment.

Guaranteed Accuracy

Who wants to employ an extract or comparison tool that cannot guarantee to assemble 100% of the related data? Any generally applicable software product for application testing must deliver 100% accuracy — no matter how many tables, relationships or different database types exist in the environment. Yet, this is a major difficulty for table-level migration products and for products with limited support for complex data relationships.

The requirement to keep many related rows synchronized throughout the testing process — all the while maintaining referential integrity — is a huge challenge for a universally applicable software product. Most simply cannot do it. And their users face concerns about accuracy and reliability.

Reusability

Quality application testing must be comprehensive, consistent and repeatable. A generalized testing toolset must allow users to define, save, share and reuse test specifications to extract, insert, update, edit and compare test data. Consistency and accuracy must be ensured. Refreshing the database should be fast and easy. These capabilities allow companies to streamline application testing with a minimum of time and effort.

Scalability

How well will a testing solution work when the dynamics of your application testing requirements change or expand? Application development and testing environments are dynamic. The testing toolset must handle any arbitrarily complex data model or the IT organization will have extra work to do. Any software technology that claims to support application testing must not impose artificial limits on the number of tables or the kinds of relationships it handles — because that puts the IT organization back into the custom coding business.

Meeting the Challenge with Relational Tools

Princeton Softech's Relational Tools™ enable IT organizations to meet even the most complex application testing challenge by providing all the fundamental components of an effective testing strategy:

- Extract referentially intact subsets of data with 100% accuracy to create realistic test databases no matter how many tables or relationships are involved.
- Insert or load subsets of related data to quickly build realistic test databases. Update or refresh the test data consistently to preserve the integrity of the test environment.
- Mask sensitive data to ensure compliance with regulatory requirements for privacy. Transform test data to meet specific test case requirements.
- Provide aged data for "time-dimensional" testing so that future events (year-end processing, revision to effective dates and so on) can be adequately tested.
- Browse and edit test data to force error conditions and resolve problems. Reviewing data in its relational business context provides a clear vision of the data model.
- Compare the images of the test data before and after exercising the application to validate expected test results and identify anomalies automatically and with pinpoint accuracy.
- Integrate test data from other database management systems (Oracle, DB2 UDB, Sybase, SQL Server and Informix).
- Integrate DB2 and legacy data into the test environment and take advantage of COBOL or PL/I copybook information for transforming legacy records.

When these capabilities are in place:

- Developers can ensure that new application functions perform as expected during unit testing and modifications do not cause problems during integration testing.
- Quality Assurance staff can ensure that the entire system operates as expected and validate that interfaces with other systems work properly.
- Business-unit users can ensure that the system meets their expectations for functionality and performance during acceptance testing.

- Database Administrators can spend less time on creating and maintaining the test environment and more time on performance tuning, backup and recovery processing and managing production databases.

What makes the Relational Tools unique?

Princeton Softech's Relationship Engine™, at the core of the Relational Tools, is a unique technology that understands and processes related data from multiple tables and ensures that each test subset is always referentially intact and logically complete.

For example, in a subset of customer data, one customer may have items that are backordered while another may not. From one customer to the next, the number of rows retrieved — from any number of tables — will vary. But the Relationship Engine always gets the right rows for the right customers — complete and intact — every time.

The Relational Tools use an active repository, the "PST Directory," to store the user-defined business rules (see Figure 1). The portion of the PST Directory that stores data model information can be populated automatically from the database or from any number of third party dictionaries. In addition, users can define relationships that exist but are not known to the DBMS.

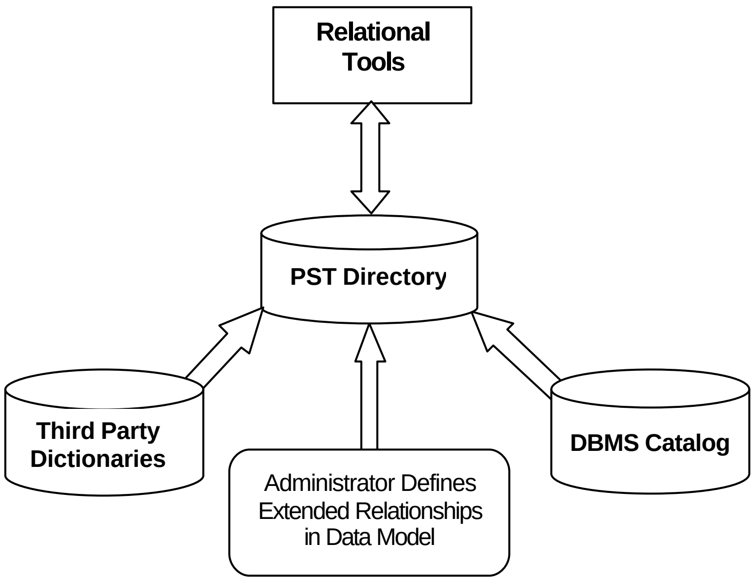

Figure 1. PST Directory Stores User-Defined Business Rules

Using the PST Directory, IT staff members can define, share and reuse different Access Definitions that represent, in essence, varying subsets of the database — related rows from many tables. Princeton Softech's Relationship Engine technology assembles all related rows from every table into a single referentially intact subset.

What do the Relational Tools offer?

The Relational Tools extend beyond extracting data, allowing users to copy, move, browse, edit and compare complete subsets of related data (see Figure 2). Organizations can design comprehensive testing strategies that include realistic test data while improving productivity and overall application quality.

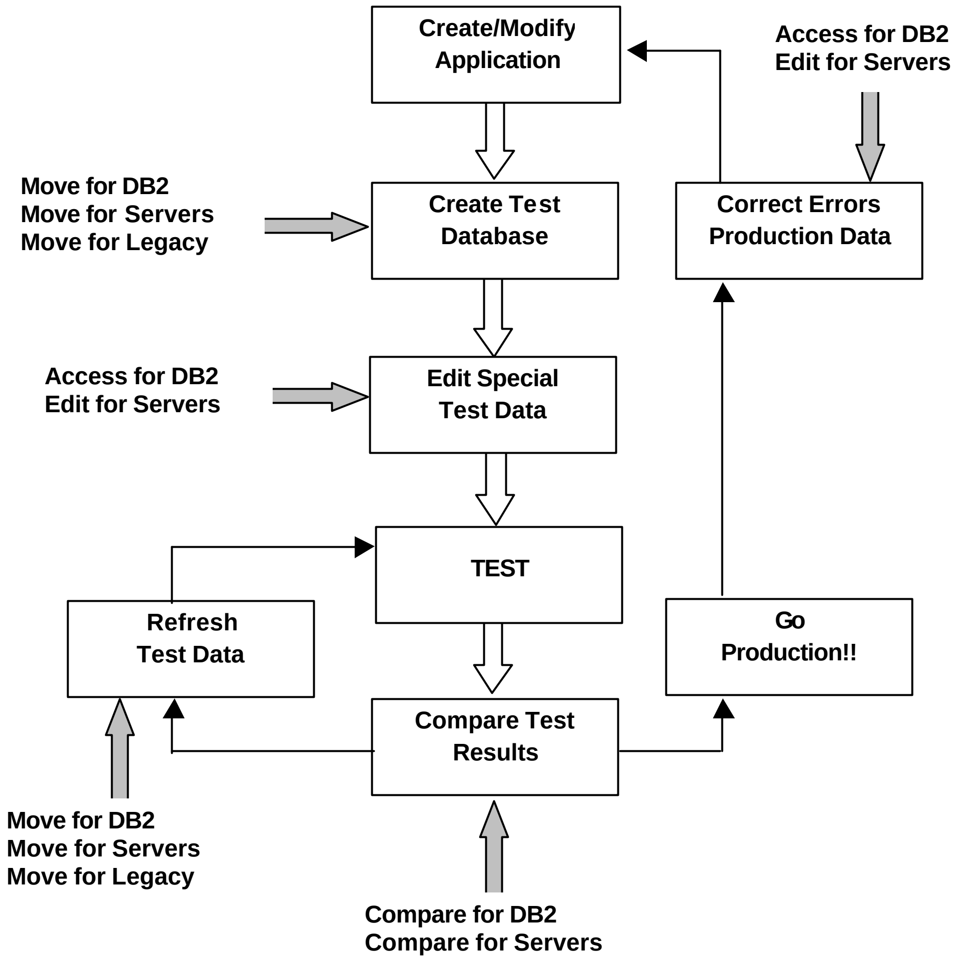

Figure 2. Testing Smarter with the Relational Tools

Test Smarter with Proven Technology

Along with the need to test and deliver reliable applications, companies that want to remain competitive must reduce the time to market and reduce the cost of quality by resolving problems before applications are deployed. IT organizations cannot risk an application testing strategy that is only 95% reliable:

- With the ability to create realistic test data that is referentially intact and 100% accurate, the testing environment has the critical element necessary for quality testing.
- With the ability to browse and edit that data in its relational context, it is easy to create special test cases that exercise every facet of an application.
- With the ability to compare test results before and after each test scenario, the differences can be identified automatically, and evaluating test results is faster and more accurate.

Why test smarter? Because it makes good business sense and can deliver a greater return on investment over the application life cycle. The Relational Tools provide proven technology and a truly universal solution to help companies streamline application testing:

- The Relational Tools for DB2™ provide capabilities to move, edit and compare relational data. Move for Legacy™ extends these capabilities to provide federated data access to legacy and DB2 data from different sources.
- The Relational Tools for Servers™ provide capabilities to move, edit and compare relational data using the industry leading database management systems, including Oracle, DB2 UDB, SQL Server, Sybase and Informix.

In addition, as long as the client connectivity is in place, it is possible to manage data across different operating systems, including Windows 95, 98, 2000, Windows NT, OS/390 and UNIX.

The Relational Tools, for both DB2 and Servers, are the only products available today with the required generalized software architecture. That's why hundreds of companies, including many of the world's top IT organizations, rely on the Relational Tools to test smarter every day.

princetonsoftech.com

111 Campus Drive
Princeton, NJ 08540-6400
Toll free 800.457.7060
Phone 609.627.5500
Fax 609.627.7799

ref no: 27543-1

princeton softech