TestBase™

SoftBase
A Fresche Solutions Brand

Db2 for z/OS Test Data Management Revolutionized

TestBase's Patented Slice Feature is an Answer to Db2 Testing Challenges

The challenge in creating realistic representative test data lies in extracting and reproducing subsets of production data, quickly and accurately. TestBase from SoftBase Systems simplifies the creation of test data for your application development environment while also satisfying your security and privacy requirements.

- Do you have too many development and test environments or too few?
- Are your development and test environments exceedingly complex?
- Do your developers spend too much time resolving test environment set up problems?
- Do your developers spend too much time doing routine test data conversion for changed tables?
- Do your DBA's spend too much time setting up and maintaining test environments and populating them for developers?
- Are your project deadlines jeopardized by your inability to test in parallel?
- Is software quality sacrificed by your inability to test in a timely way?
- Is regression testing so difficult it's now impractical?

These are some of the problems the TestBase Slice feature addresses. This patented technology allows each programmer or tester to manage their own slice of Db2 test data, dramatically simplifying program testing and making true regression testing practical. Each slice can be maintained independently of others within the same tables. Slice encourages better testing by reducing the effort needed to create, manage, and use a test environment.

Background

Application testing requires a methodology that can traverse the testing process. In real-world projects, different components of the system will be exercised in different stages of testing. However, Db2 application testing often demands working with an additional myriad of testing challenges. We are all aware that each stage of the testing process has varying degrees of complexities and demands on coordinating testing efforts. For example, during unit testing, tests are performed on individual programs to determine if they meet defined specifications. Individuals working alone, typically a programmer, perform unit tests. Very limited coordination is necessary for this type of testing. String testing advances the process with a series of programs to confirm that they communicate necessary information to each other. Then, the testing process advances to the system level, where business cycles that the system was designed to meet increase the complexities of managing and coordinating all of the activities and people associated with that effort. Of course, application modifications will be made and regression testing starts its revalidation processes. Tests will be performed on a system to determine if it produces the same results or expected new results after a change. The regression test concept is that you have tested some code and assert that it has passed the tests. However, the application is constantly changing due to enhancements and fixes. Any changes have the potential of invalidating previous tests. There are additional kinds of testing such as performance, stress, production simulation, and parallel production testing done prior to an applications' implementation. However, this document only addresses the Db2 testing or challenges specific to unit, string, system and regression testing.

Testing Challenges

Each type of software testing presents its own sets of challenges. Some common ones include the development of test data for individualized branches of the program being unit tested. In other words, test data that meets each of the branch conditions. This implies that each program requires its own set of test data. Once a test has been performed, program changes are usually made to correct problems and the test is repeated. It can be difficult to keep track of the test data necessary to repeat the tests and re-establish the test data especially when test s require isolation or a predefined order to the testing schedule.

Test Data Isolation or Scheduling

Different programs serve different functions and may need to be isolated or scheduled. Some examples might include purge programs, report programs, and update programs. Testing a report program while testing the update program may produce unpredictable results. Were the reports wrong due to errors in the report program or just the fact that the update program changed the data before it was reported? Did the update program fail because the purge program removed the data before it could be updated? These can be time consuming and frustrating questions to answer. In real-world projects, different components of the system will be in different stages of testing. For example, some programs might still be in unit test while others are in string test. Allowing more than one group of testers to perform these tests in the same environment on shared data is neither practical nor advised. Different test activities can adversely affect one another. Testing in this way is very error-prone and requires extensive coordination that will slow project progress and impede if not prevent, parallel testing.

Test Data Verification

Individual unit tests also have the problem of determining whether the program functioned as expected. Reports and screens are easy to verify, but did the data stored go through the expected transformation. Did the employee given the 10% raise have his salary updated by the correct amount in the data store? Was some other employee attribute accidentally updated at the same time? A related problem is the reporting and documentation of the test. Screen prints are expensive to collect and don't show a complete picture. It would be nice to force each unit tester to state what he wants changed in the data and then produce a report showing that the desired changes, and only the desired changes were made.

Additional Db2 Challenges

Using a database management system like Db2 can provide additional challenges including:

Db2 Locking

Db2 uses a lock mechanism to assure that readers of information get accurate data and that only one application can update at a time. Locks can be taken at the dataset or tablespace, the page, or the individual row. This is referred to as lock granularity and is determined by the lock size of the tablespace and the isolation level of the bind (compilation of SQL). Typically, a locksize of page is chosen because row locking has a fair amount of overhead. Using this locking strategy, if two testers have data that happens to be on the same page, they could prevent each other from testing. To make matters worse, locks are not released until commit. On-line debugging facilities can hold locks for hours preventing one or more tests from being able to execute. In short, the very facilities that guarantee production data integrity make testing a challenge.

Db2 Utility Processing

The Db2 load utility or its replacements are often used to re-establish test data. These loads have to be done table by table and JCL built to accomplish the task. If this is not bad enough, while the load utility is executing, the tables are unavailable for other testing work. After loads are done, the tablespaces may require image copies or additional utility operations such as the check data utility to make the data available again.

Db2 Referential Integrity

Referential integrity constraints can cause additional consideration as to how the data is saved and loaded or even which tables are required for testing. Although these constraints are beneficial to the integrity of the application, they will require data in other tables, and consideration in the data loading for test purposes.

For Information: Call: 800.669.7076 | E-mail: sbsales@freschesolutions.com **| Visit:** www.softbase.com

Db2 Table Changes

Once test data has been developed, when table changes occur, the test data has to be converted. The conversion effort is related to the amount of change, how much test data has been developed, and how many environments must be changed.

Conventional Db2 Application Testing Solutions

The conventional solutions to these challenges each have their own drawbacks. They generally involve using SQL or physical sequential files to develop and maintain test data and scheduling or creating additional environments (multiple copies of the same tables) or a combination of both to perform the tests. SQL delete statements can be created to clear the tables. One delete statement per table must be coded and executed prior to each test. Additional SQL INSERT statements can be coded and executed to build the test data required. One set of inserts per table.

Physical Sequential Files

Once data has been inserted, it can be unloaded to a physical sequential dataset, again one per table. JCL must be written to unload, and additional JCL written to re-load. This technique can be used to avoid the delete problem. It has the drawback of making each table unavailable while it is being loaded.

Compare Utility

If data is unloaded to the physical sequential files – one per table, a compare utility can be used to verify that data has changed as expected or not changed in the case of regression testing. This provides some automation of the test verification. JCL must be built to do the unloads and compares – again one per table. The datasets containing the unloads must be maintained for comparison purposes. Data must be converted to display manually and sorted by the primary key for each table. Preferably, the primary key would be the lead part of the unload records; otherwise most compare facilities will not match the proper records.

Scheduling

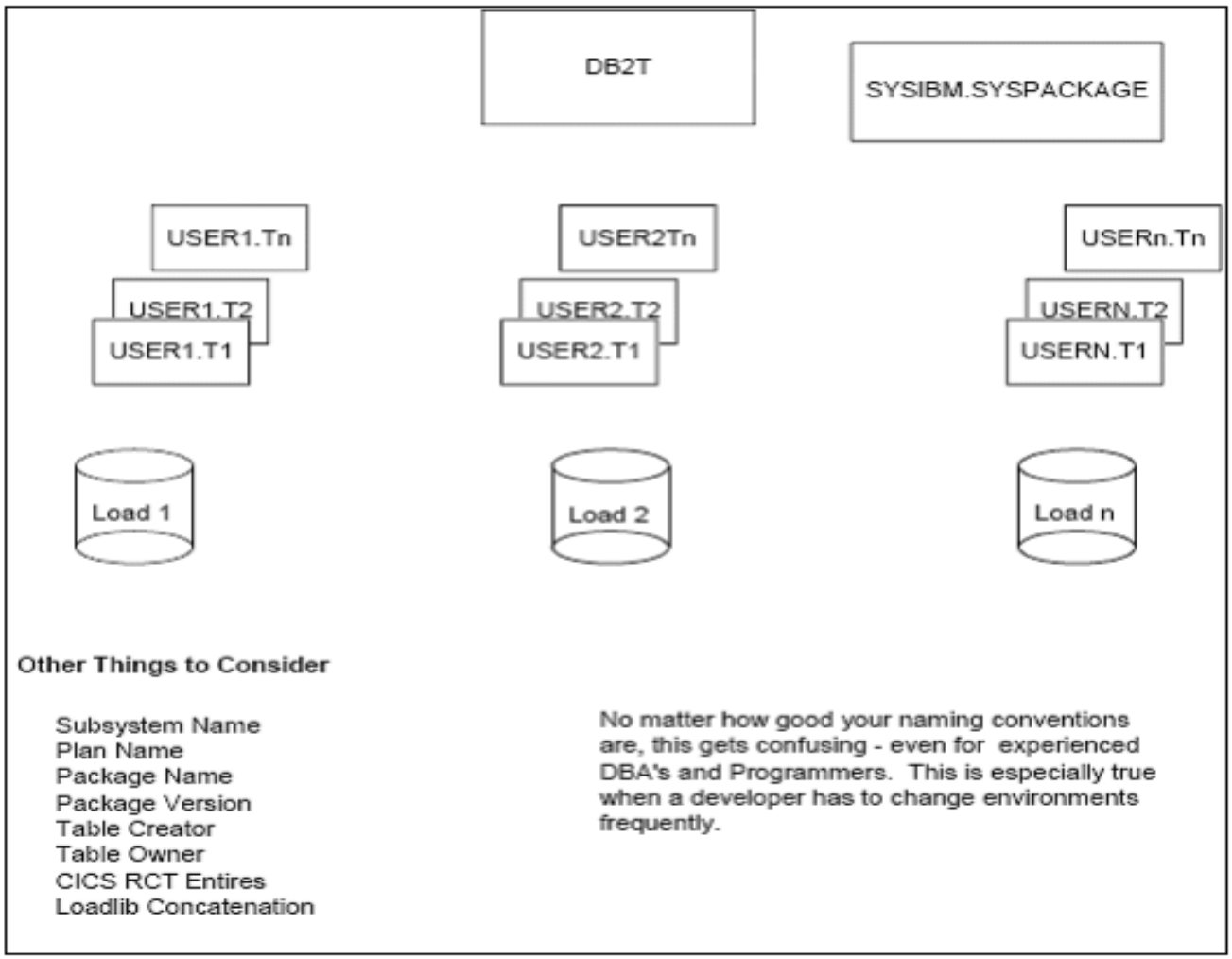
By far the easiest solution to test execution is to simply schedule testing. Monday is the report program, Tuesday is the update program, Wednesday is the Purge program, etc. Monday is string test #1; Tuesday is string test #2. Monday is system test #1 and Tuesday is regression test #1. The problem is time. Given enough time, we could proceed this way rather easily.

Scheduling and Key Assignment

If we assign key values to various test processes, we should be able to allow at least some testing to happen in parallel. The report program test will use employees 100 through 200. The purge program will use employees 200 through 300, etc. Oops, what about the department table? It is keyed by department number. All employees in the 100 through 200 ranges must be assigned to departments 10 through 20 and employees in the 200 through 300 ranges must be assigned to departments 20 through 30. The drawbacks here are that each additional key must be assigned a range for each test group and the relationships propagated properly. Even so, errors can occur in programs that will occasionally cause updates to data that is not intended. Db2 lock contention can still happen despite careful choices of key assignments.

Multiple Copies of Tables

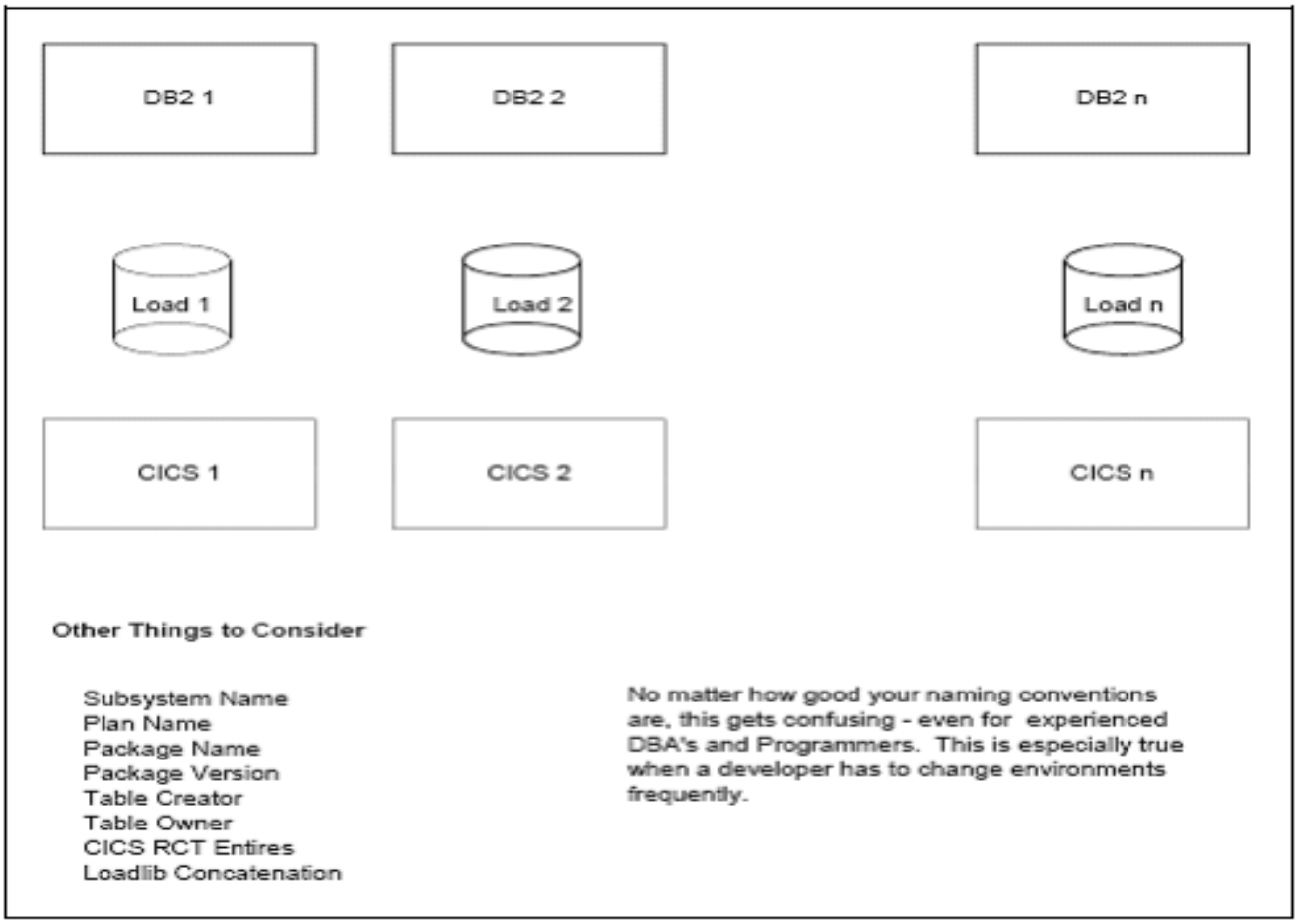
One sure way to avoid lock contention is to give each tester a set of tables. Such an environment might look like:



This avoids the entire Db2 load, and lock contention problems entirely. It is an expensive alternative. We've seen shops with only 400 tables grow to more than 8,000 tables! Soon, even with Db2 alter and migrate tools, this alternative becomes the full employment act for Db2 DBA's. Application programs each must be bound properly – using the correct Db2 table creator. The same program can be bound many times in such an environment and can lead to huge SYSIBM.SYSPACKAGE sizes. Suppose you have a common module that is used in many functions and need to make a change to it. It should be bound numerous times and tested everywhere. Another risk of this solution is not properly making a table change to all environments. This could result in code being developed against an outdated table definition. In general, a lot of coordination is required for database changes with this approach. The more environments and change, the more coordination is required.

Multiple Db2 Subsystems

This solution is an even more expensive variation of using multiple copies of tables. The advantage here is that Db2 creators do not have to be changed. Such a solution might look like:



Modify Programs and Add Additional Column(s) to Tables

One vendor has a product that reads program source and adds a “where clause” to each SQL query to point at a specific data value. This minimizes the number of environments and usually solves lock contention problems. Unfortunately, it requires modification of program source and that several coding conventions be adopted to set the additional columns properly on inserts. Preprocessors must be written and executed to add the where clauses to the SQL. When SQL syntax changes are made, the pre-processor must also change to recognize the new syntax.

A New Approach: TestBase Slice

The need for test case isolation has been established. However, it has also been established that there are considerable problems with the common methods of providing this isolation. TestBase Slice provides isolation without these problems and provides a set of integrated tools for managing and comparing the test data. Each slice is completely isolated from the activities of other testers. Through a unique strategy, the Slice Tool provides this isolation without modification of program code. The important point is one set of tables in a single subsystem provides concurrent testing capabilities insured by Db2 to be independent.

Slice integrated functionality includes the capability to:

- Build and maintain test data in a friendlier way than modifying saved SQL statements.
- Unload test data from all tables in a system in one command with minimal lock contention for other testers and users.
- Load test data back with minimal lock contention for other testers and users and without making tables unavailable because of the load utility limitations. If data formats have changed, it should convert to the new data format wherever possible. New columns should be defaulted and deleted columns ignored. Changes in the Db2 referential integrity definition should not prevent valid data from reloading.
- Catalog and inventory the test data unloads.
- Allow users to test with total independence – purge testing in parallel with report testing in parallel with update program testing.
- Provide comparison facilities that show the keys of rows inserted, updated, or deleted in display format and changed data values new and old in display format.
- Allow users to share copies of test data.

TestBase Slice provides benefits in many areas of Db2 application testing:

- Unit Db2 Slice allows users to share or isolate unit test data within a single set of Db2 tables. This provides each programmer with one or more “slices” of test data, each tailored to provide repeatable tests for their program or programs. Data can also be copied and shared by programmers, eliminating the tedious creation and recreation of test cases necessary to verify that a program is functioning as expected.
- String Db2 Slice allows for multiple string tests that can be executed concurrently.
- System Db2 Slice allows concurrent system testing. Normally performed in units called phases or artificial “test days”. Each phase or test day builds upon the previous. Typically, system testing is linear–day one testing must be completed before day two testing can begin. Day 2 must be completed before day 3 can begin, etc. Using Db2 Slice, not only can day 1 testing be performed concurrently with day 2 testing, but it can also be performed within the same set of Db2 tables.
- Regression Db2 Slice dramatically simplifies the maintenance and verification steps used in a regression test. Using Db2 Slice allows for more thorough regression testing which is often underutilized method of testing due to the complexity of creating and maintaining test data

Summary

With its Patented Slice feature, TestBase is one of the most powerful tools available for companies involved in creating, testing, and implementing key business applications. It is another step in providing required test data management capabilities.