

Beyond the GUI: What You Need to Know about Database Testing

Presented by: Mary R.Sweeney

***Exceed Technical Training
& Consultation***

Copyright Sammamish
Software Services 2003. All
rights reserved.

1

Today's complex software systems access heterogeneous data from a variety of backend databases. The intricate mix of client-server and Web-enabled database applications are extremely difficult to test productively. Testing at the data access layer is the point at which your application communicates with the database. Tests at this level are vital to improve not only your overall test strategy, but also your product's quality. In this presentation you'll find out what you need to know to test the SQL database engine, stored procedures, and data views. Find out how to design effective automated tests that exercise the complete database layer of your applications. You'll learn about the most common and vexing defects related to SQL databases and the best tools available to support your testing efforts.

The Data Access Layer

- Testing at the data access layer is the point at which your application communicates with the database.
- In this presentation we'll discuss why tests at this level are vital to improve not only your overall test strategy, but also your product's quality

Copyright Sammamish Software Services 2003. All rights reserved.

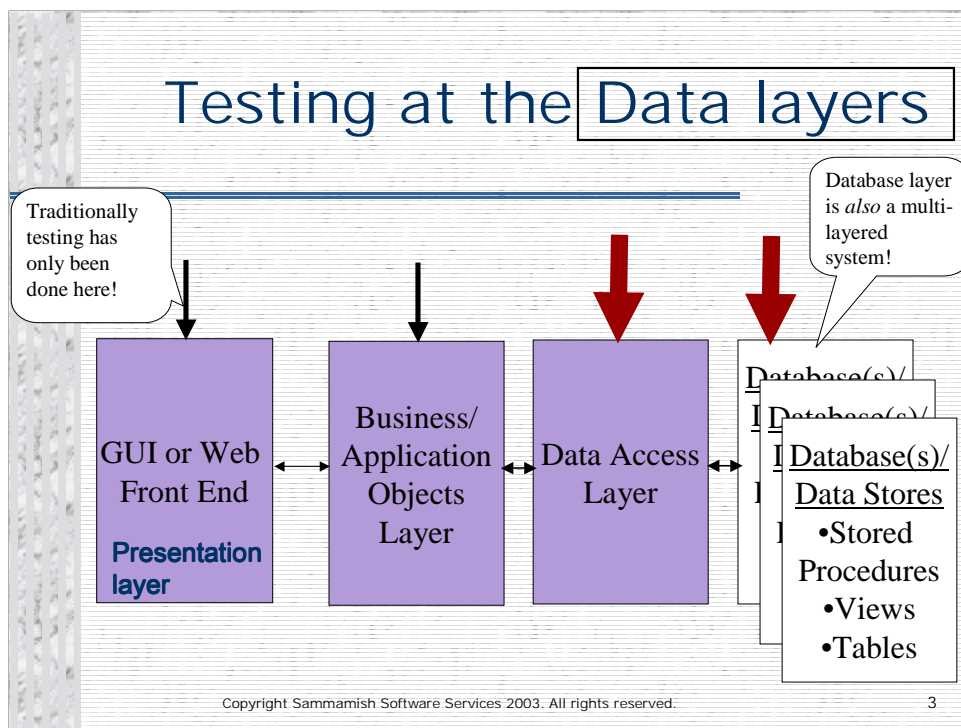
2

How to design effective automated tests that exercise the complete database layer of your applications. Donation Street demo

The most common and vexing defects related to SQL database

The best tools available to support your testing efforts.

A strong foundation in Structured Query Language is becoming increasingly necessary and even required by some companies in today's data environments. Many test professionals are being asked to bridge the gap between database administrators and database developers by testing the integrity between the database application and it's data stores. Understanding the primary database manipulation language – Structured Query Language -- for Relational databases is key to being able to perform effectively in this arena.



Most applications are data-driven in some way in today's environments. The complex systems we deal with have many different layers. Testing experience has found that we cannot effectively test the application without being aware of these layers and what issues and bugs they can introduce into the system. A data-based application has a data access layer which is the code written to manipulate and perform actions on the database. The databases and data stores used by the application present a multi-layered system in and of itself.

A given database may have to allow for more than one client application, for example, an application for internet user and those for intranet or client/server applications, perhaps also an administrative application. These additional client applications may use the database in unpredictable ways. Certainly an "advanced user" application or even a DBA going in behind the scenes could touch the data in unforeseen and potentially harmful ways. Hence the database component may need to be tested for uses beyond those of the current application and further it should be regularly checked for data integrity and consistency. This is particularly true if, for scalability reasons, the business rules are not kept on the database backend but in the database application front-end (presentation layer) and/or middle-tier layers.

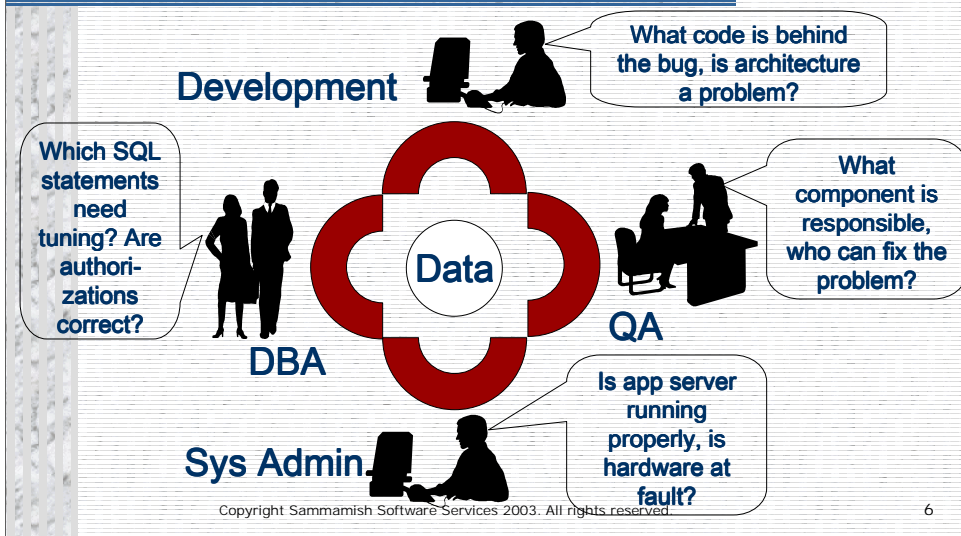
Why testing at the data layers is important

- Today's **complex** software systems access heterogeneous data from a variety of backend databases.
 - Corporate acquisitions make multiple database backends the norm
 - Focus is on **functionality, reliability, recoverability, capacity planning and scalability, performance, data accuracy**
- The intricate mix of client-server and Web-enabled database applications are difficult to test productively.

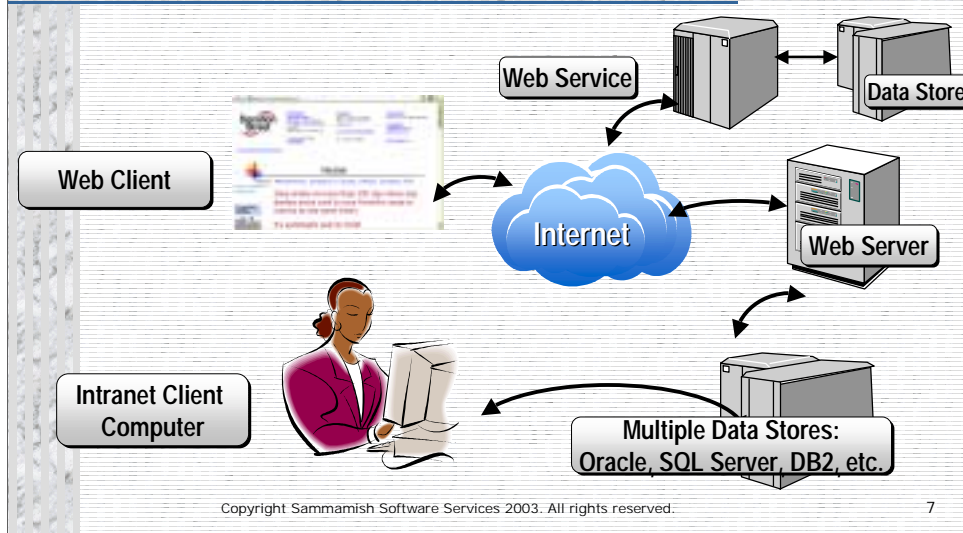
Yes, Data must pass QA too!

- All data, especially production data, should be QA'd with periodic sanity checks
 - **Is the data: Reliable, reasonable, consistent, accurate?**
 - Ensuring this is a QA function
 - Who's actually doing this?
 - Frequently left to DBA's who may not consider QA part of their job (or a small subset)

QA'ing data; who's responsible?



Complexity of database interaction



Why test database objects?

- If you don't test DB objects, stored procedures and views, **you're missing critical application functionality**
- Increasingly stored procedures and views are used on the database backend in modern applications
- Application functionality is moved to the database, **Why?**
 - Performance optimization by the DBMS
 - Security: access can be limited
 - Robustness against hacks

Copyright Sammamish Software Services 2003. All rights reserved.

8

Since you're a tester, you always ask "why"? Why am I here? What's the point?

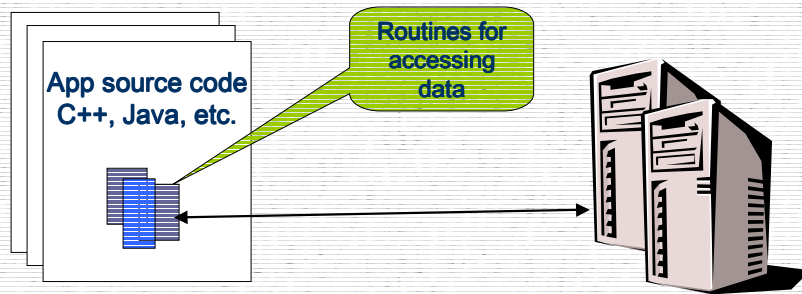
With the increasing use of data on the internet and in every application, the need for skilled testers in databases is growing quickly. Stored procedures are increasingly used by experienced database developers to ease the complexity of calls to the database backend. These procedures contain critical tasks such as inserting customer records and sales. They need to be tested at several levels. Black box testing of the front-end application is important, but makes it difficult to isolate the problem. Testing at the backend can increase the robustness of the data.

Also a given database may have to allow for more than one client application, for example, an application for internet user and those for intranet or client/server applications, perhaps also an administrative application. These additional client applications may use the database in unpredictable ways. Certainly an "advanced user" application or even a DBA going in behind the scenes could touch the data in unforeseen and potentially harmful ways. Hence the database component may need to be tested for uses beyond those of the current application and further it should be regularly checked for data integrity and consistency. This is particularly true if, for scalability reasons, the business rules are not kept on the database backend but in the database application front-end (presentation layer) and/or middle-tier layers.

Don't the major test tool vendors cover this? Not that I've seen! Sorry! I don't want to name names here but we all know who they are. They seem to have enough to do as it is providing the support for GUI testing and performance. Testing database objects can be done by pretty much all the tools in some limited fashion, but you don't necessarily have all the flexibility you need in the IDE of these tools and therefore have to rely on coding using their macro language anyway. As much as possible, I recommend bypassing their programming macro language and writing your tests in SQL script code directly in your RDBMS or using a variety of relatively simple scripting languages – VBScript, PERL, Ruby, Javascript. Why, you ask again? Because they're more reusable and portable in terms of your database, of course, and also because the skill set for using these languages will be more available in the personnel you're able to hire, train, and retain. I can think of other reasons too, but those are pretty good, don't you think?

Traditional application architecture for data access

- Data access routines reside within the application source

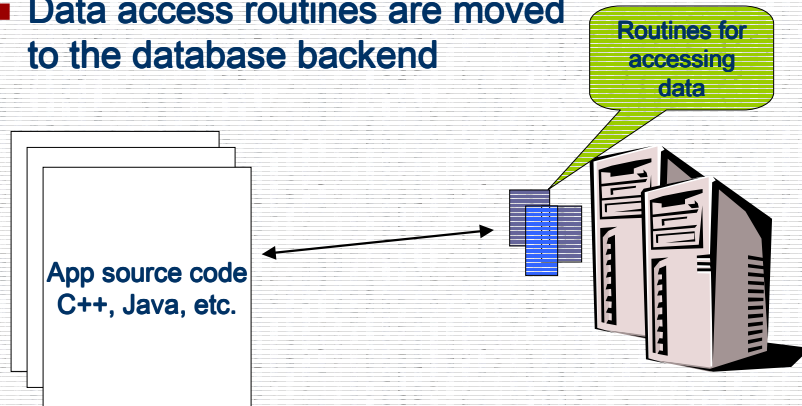


Copyright Sammamish Software Services 2003. All rights reserved.

9

Modern application architecture for data access

- Data access routines are moved to the database backend



Copyright Sammamish Software Services 2003. All rights reserved.

10

What kinds of tests?

- A *lot* of tools available focus on performance and scalability. This is good! However...
- Tests must go beyond performance tests and **also** incorporate critical functionality testing
 - testing of database objects such as stored procedures and views isolates critical areas
- Don't developer's do this?
 - Dev's do unit tests to ensure basic functionality
 - QA adds tests to break the system!

Common defects in SQL databases

- data corruption
 - frequently due to poor design or design that doesn't scale
- redundant data –
 - common error for developers not sufficiently skilled in RDBMS
 - Don't assume developers have sufficient design skills!
 - Test Engineers must have RDBMS skills to pick up these types of errors

Copyright Sammamish Software Services 2003. All rights reserved.

12

Common defects in SQL databases

- Inconsistent data
 - Multiple, unpredictable clients
 - Not consistently following business rules within *all* applications that touch the data
- Redundant validation
 - Validating business rules on the db, as well as the client, can cause conflicts if they're not well-coordinated

Copyright Sammamish Software Services 2003. All rights reserved.

13

***reasonableness of the data.** If you regularly receive a certain amount of sales, an influx of rows added to critical system tables well over the norm should raise a red flag. So should lack of records on days that should have generated a regular amount of sales.

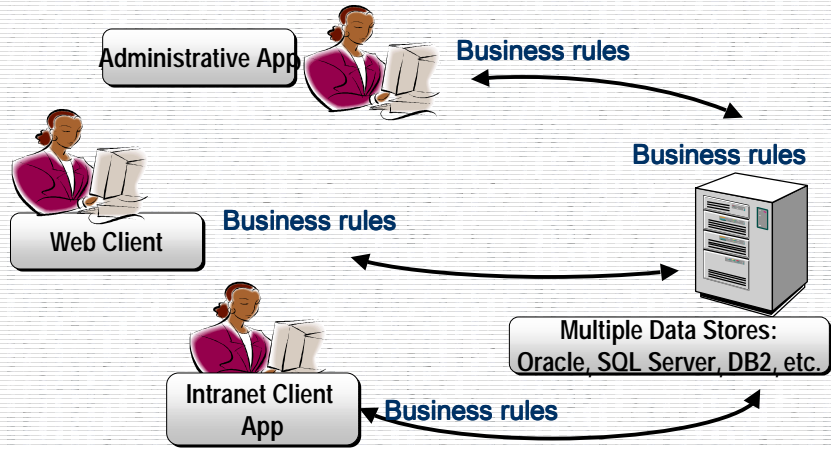
***data corruption.** Data should be regularly QA'd to find garbage data within columns

***redundant data.** Hidden duplicate records (same customer added twice with different primary keys). How many of us get multiple copies of newsletters? This annoys and loses customers.

***evidence of hacking.** Plan test cases to uncover multiple similar records added on the same day.

***inconsistent data.** Data records added to the same database through multiple applications can add inconsistent data. For example, if the customer application adds data with formatting applied to columns such as phone numbers and e-mail, but the Admin application to the same data does not, then the data can become inconsistent and not match properly with reports. A worse example is required fields set at the customer application not being set at the Admin application leaving orphan records or unintelligible data.

Multiple clients



Copyright Sammamish Software Services 2003. All rights reserved.

14

More Common defects in SQL databases

- Evidence of hacking
 - SQL Injection attacks
 - Malicious attempts to corrupt data
 - Serious attempts to reprice data for theft or embezzlement
 - Multiple users added
 - Denial of service attempts
 - Overloaded tables
- **Conclusion:** We need to set up test cases to determine the health of the data

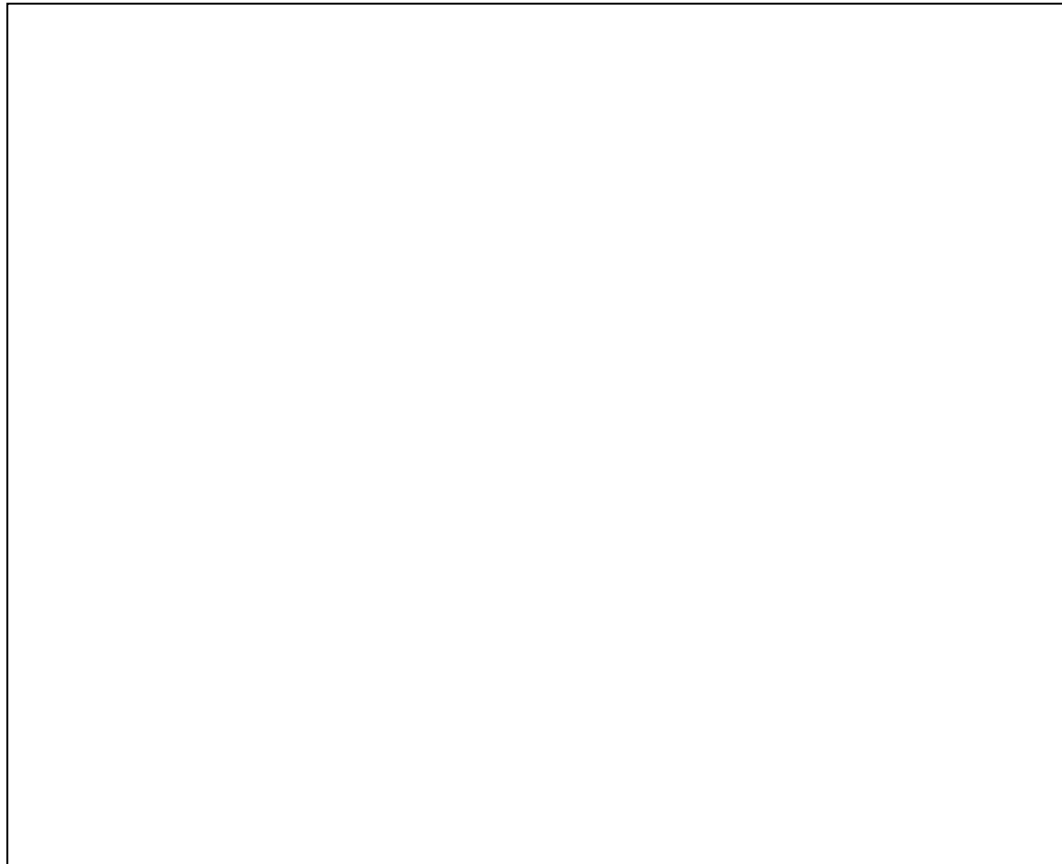
QA

Back end vs. Front End Testing

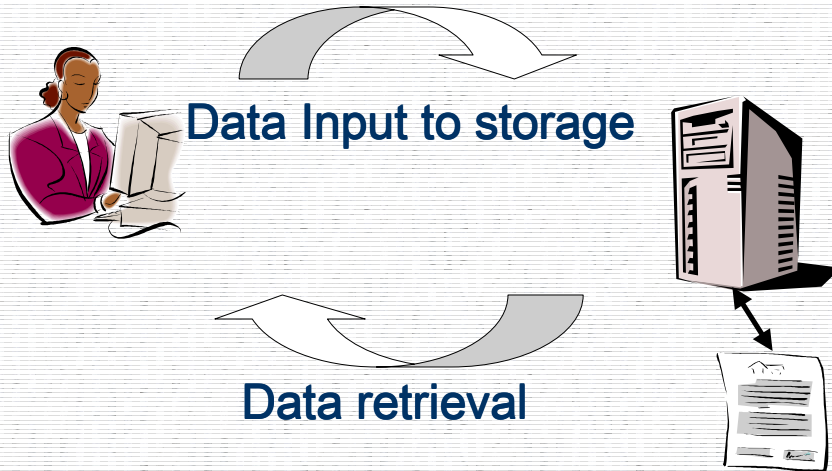
- Traditionally most data testing is done at the GUI
- Why Test at the database backend?
- It's not easy to present test cases from the front end that determine the complete health of the data at the database backend
 - Corruption can occur at **any** of the layers
 - **We must present verification of application correctness as data travels through system**

Copyright Sammamish Software Services 2003. All rights reserved.

16



Examining the data's round trip through the app



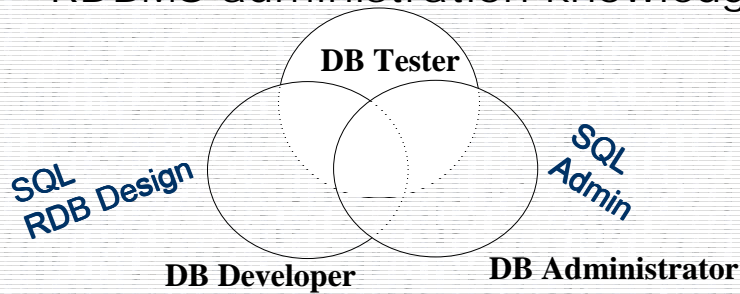
Copyright Sammamish Software Services 2003. All rights reserved.

17

Data corruption can be introduced at either of these round trips of the data through the application and in many locations along the way as the data passes through many security gates and servers. It's the tester's responsibility to verify the path of the data through every phase of both of these legs of the trip. This round trip can include passing through many security gates and servers.

What DB Basics does your test team need?

- Relational Database Design fundamentals
- SQL fundamentals
- RDBMS administration knowledge



Copyright Sammamish Software Services 2003. All rights reserved.

18

Why should test professionals know DB Basics?

- Not *all* members of a test team need to know it
 - Need at least one person on the team who is DBA level
- Those team members assigned to verify data transformations within the application **MUST** know it!
 - Can look for and find more common design problems
 - Can speak intelligently about how to address/fix data issues

Copyright Sammamish Software Services 2003. All rights reserved.

19

A common misconception in testing is that you can test and uncover data issues within an application by ignoring the backend and testing only on the front-end of the system. However, this approach ignores the layers we discussed earlier and makes it more difficult to isolate the bugs found. Today's testers must be sharper and more knowledgeable.

Why should test professionals know DB Basics?

- Isn't this a development task?
 - Isn't this the same as Unit testing?
 - QA does not and should not replace good unit and integration testing strategy by development
- Development tests are usually built to *ensure functionality* and do not typically incorporate full QA, ie., a full test of the component
 - Example: Development harness accesses component and executes basic methods *but* doesn't do any testing of valid or invalid input, equivalence class testing, boundary analysis, etc.

Copyright Sammamish Software Services 2003. All rights reserved.

20

Why should test professionals know RDBMS Design basics?

- Inadequate knowledge of Relational db design fundamentals leads to logic errors and very common bugs in systems
 - Basic normalization principles can and should be tested but isn't -- because most testers have no idea what that is!
- Effective DB testers should be able to uncover design problems quickly
- Able to recognize neophyte developer errors easily
 - A test pro with DB background can uncover poor design bugs by adding a number of basic tests

Copyright Sammamish Software Services 2003. All rights reserved.

21

The intent of this course is to focus on SQL in a test environment but we will review DB design basics since lack of understanding of these essentials can lead to common problems in an application. The test professional cannot expect to be proficient in SQL yet ignorant of relational DB design fundamentals.

Why should test professionals know SQL Basics?

- SQL statements can be used to:
 - Verify SQL statements used within the app
 - Insert and remove test data
 - Verify existence of data integrity on the DB
 - Reveal corrupt data
 - Duplicate data
 - Orphan records
 - Create and run automated tests in the DB backend
 - Create test cases to protect against common DB hacks; *we must be as smart as hackers!*

Copyright Sammamish Software Services 2003. All rights reserved.

22

If you're testing an application which has the primary focus of moving data between users and relational data stores, then knowledge of Structured Query Language is essential to be able to efficiently and successfully test the application's use of data. The list above of possible applications is merely the short list. There many possible uses for SQL in a test environment. In the following section we will study SQL essentials from the perspective of the Test Professional.

Administration issues for database testing

- Server-side management issues:
 - Defective Installations and Configurations
 - Database Security Issues
 - Disaster Recovery Issues
 - Creating a Test Server
 - Distributed Data Issues
 - Replication Issues
 - Reliability and Capacity planning
- A tester with some DBA background can add QA in these important areas

Copyright Sammamish Software Services 2003. All rights reserved.

23

In addition to the areas we've mentioned so far in this class, above is a list of further issues to consider in a database test project. The answer is not always SQL but the Test professional must be able to know when and where the SQL implementation for the DBMS being used can support these test areas.

How to design effective tests to exercise DB layer

- Test first *works*
 - It is possible, and desirable to design tests for the data layer prior to or in conjunction with *creating* the data layer
 - Developers create the harnesses that exercise the business objects and database layers
 - Testers fill in the tests using standard techniques for uncovering problems,
 - Data driven tests
 - Equivalence class and boundary analysis, etc.

Copyright Sammamish Software Services 2003. All rights reserved.

24

How to design effective tests to exercise DB layer

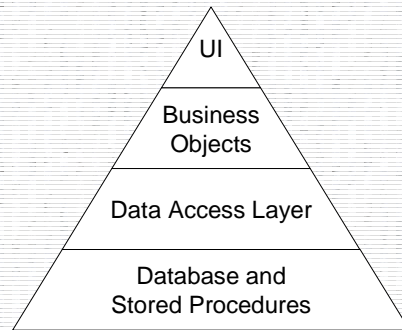
- Case Study: Donation Street
 - Incorporated test *first* strategy (mostly)
 - Harnesses to test database objects are created as the Data Access layer is created



25

Donation Street Architecture #64

- This design eliminates the dependence of the UI on the database.
- Each layer has visibility only to the layer directly below it.



Copyright Sammamish Software Services 2003. All rights reserved.

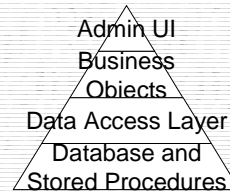
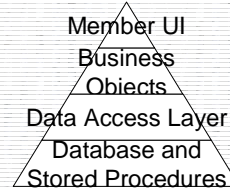
26

Donation Street began with a much more standard client-server architecture. The design model changed many times. By the time the model displayed on the slide was reached, the design had reached it's 64th iteration!

Part of beauty of this model is the multiple layers that shield the database from the UI. The UI talks only to the business objects layer so multiple UI's can be made reusing all of the same code. Tests were built into this layer that can exercise the business objects and data access layers efficiently. This method allows the testing to separately test these components and eliminate them as issues during UI testing. The real beauty is the ability to reuse the code so that code development and test development can be simply performed.

Donation Street: Architecture

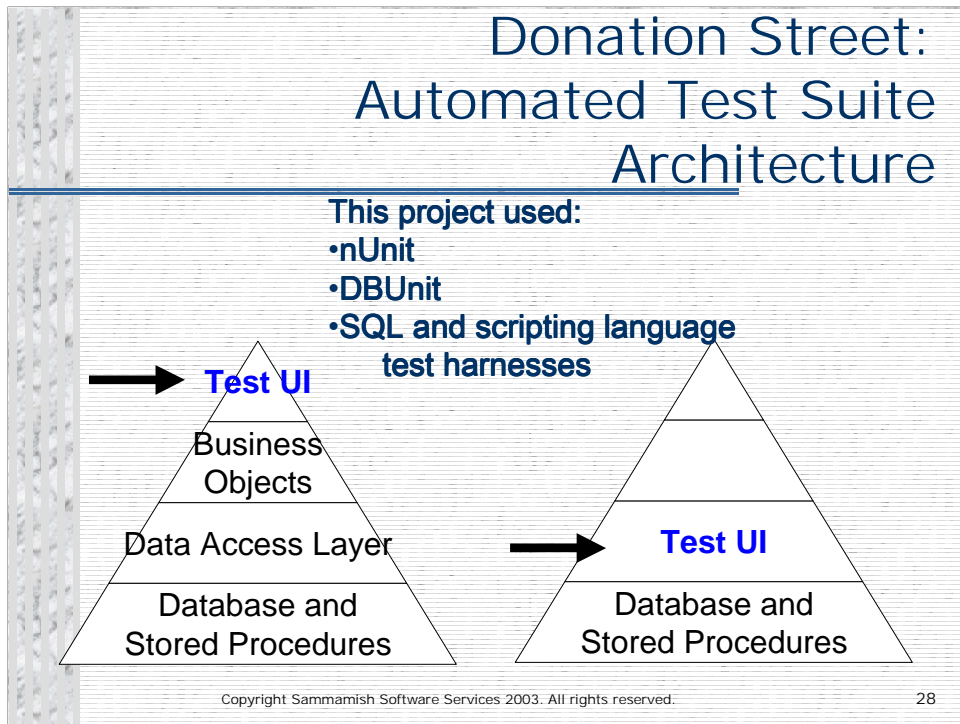
- This made it very simple to split the UI into two applications as well as to allow for more.
- Except for their UI, the applications share all the same components.



Copyright Sammamish Software Services 2003. All rights reserved.

27

Once you've created classes and objects, you can add a front end that's either web based or windows based that accesses the same objects. That's what's been done here.



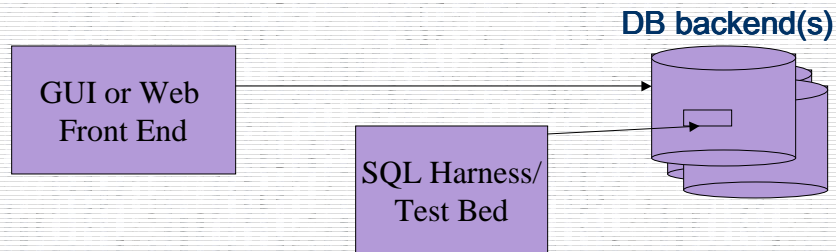
The automated tests, written in the application's software (Microsoft .Net platform) makes it easy for the developer to create test harnesses for the application's layers. It's possible to create automated tests at each layer. This particular test architecture creates full automated tests of the application's business objects, bypassing the GUI, and also includes a separate layer that independently exercises the database and stored procedures layers (multiple database backend architecture).

Down to specifics

- How do you use SQL in a test environment?
 - Executing specific queries to [analyze the health of the data](#)
 - Adding more sophisticated stored procedures to run regular tests, both during development and production
 - Adding SQL statements within programming/scripting languages to bypass the GUI
 - Examples coming up!

Back End Testing of Stored procedures

- Set up a test harness/ test bed which bypasses the front End
 - Directly accesses DB layer
 - And add harnesses for application objects comprising Data Access layer



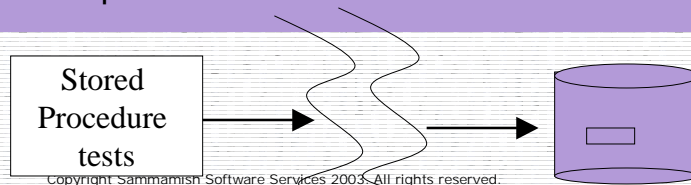
Copyright Sammamish Software Services 2003. All rights reserved.

30

Setting up a set of routines using scripts and other stored procedures, should be done to create a test bed or a more formal harness to test basic functionality. This can and should be used in conjunction with other tools, such as NUnit/JUnit, or the higher end tools, to create a full featured test environment.

Creating and Using Stored Procedures for Testing

- Can be stored within the target database or within a linked database
- Can be called from an entirely separate application (test harness)
- Testing a stored procedure *using* a SQL stored procedure.



The illustration is meant to convey that the distance between the test scripts and the backend is immaterial. The test stored procedures can be called from a separate application entirely. Although the test procedures will be stored within the target database, or possibly from a linked server.

The slides previous to this did not mention using stored procedures themselves to test database objects including other stored procedures and views, etc. Following along with our earlier example for the `usp_ValidateUser` stored procedure (to validate users attempting to access a database) another stored procedure to call that routine could easily be created rather than a separate script. This test procedure, created within the RDBMS SQL Code could use a loop to continuously read from a table of information and call the stored procedure using each row from the test data table submitted to the procedure as input.

Note: In general, this recommendation should work for any RDBMS that uses ANSI-STD SQL.

Test case: SQL Injection Attacks

ABC Corp. Login Form:

Username:

Password:

Turns this query:

Select username from user where username = 'someuser'
and pass = 'somepass'

Into this query:

Select username from user where username = '' or 1 = 1;
drop table user; -- and pass = ''

Copyright Sammamish Software Services 2003. All rights reserved.

32

SQL injection attacks can be particularly insidious. They can range from those trying to get passwords into secure systems for personal gain to those that are intentionally malicious. An excellent article about these attacks can be found at [DevArticles.com](http://www.devarticles.com) <http://www.devarticles.com/art/1/138>. Of course the referenced article talks about what they are and how to prevent them which is important information for us, of course, but our job is to write tests to ensure that these attacks have been properly prevented. In other words we write tests to simulate a SQL-Injection attack and determine how the application handles them. It should go without saying that this needs to be performed on a test database!

Using Stored Procedures in a test environment

- You can create stored procedures within the DB as testware to:
 - load test data into tables
 - remove/clean up test data
 - directly test other database objects such as views, stored procedures and constraints



Basic T-SQL stored procedure test (very basic)

```
set nocount on
select 'Starting Tests: ',
current_timestamp;
delete commercial_property;
exec inputCommercial2 10, 'TestProp1',
'Test Description1', 22;
select * from Commercial_Property;
select 'Ending Tests: ',
current_timestamp;
set nocount off
```

Copyright Sammamish Software Services 2003. All rights reserved.

34

Using scripting languages in a test environment

- Scripting languages can be effectively utilized to exercise stored procedures and other DB objects.
 - Perl
 - VBScript
 - Ruby
 - Python
 - Tcl
 - REXX
 - Etc.

Copyright Sammamish Software Services 2003. All rights reserved.

35

Advantages to using scripting languages for automated tests include the fact that they have a light footprint, i.e., are easy on the test system. Of course they can also directly emulate the calls being used by the application, especially if you use the same scripting language as the application. Be careful to avoid replicating application development, of course. Usually that's not an issue because the test scripts are smaller, more focused and therefore are able to isolate bugs better than using the application to do the test. For this reason, I advocate bypassing the GUI for certain important tests and using scripting languages to exercise the object.

The downside to all of this is, of course, that using scripting languages effectively requires more than a passing knowledge of how to write code. The IDE's for major programming languages providing debugging support, a help system and nice color coded screens, whereas you're on your own – for the most part -- using a scripting language. I don't believe coding in scripting languages always requires advanced programming knowledge on the part of the test engineer but, let's face it, it certainly doesn't hurt. These days, there are more technically capable testers available, as well.

Testing with Perl

```
<Job ID="PerlTest1">
<script language=PerlScript runat=server>
my $conn =
    $Wscript->CreateObject('ADODB.Connection');
    $conn->Open('NWDSn');
    if($conn->{State} == 1) {
        $Wscript->Echo("Connection Successful!");
    }
    else {$Wscript->Echo("Connection Failed");}
my $adOpenKeySet_CursorType = 1;
my $rst = $Wscript->CreateObject('ADODB.Recordset');
my $rst2 = $Wscript->CreateObject('ADODB.Recordset');
$rst->Open('SELECT * FROM TestData', $conn,
    $adOpenKeySet_CursorType);
$Wscript->Echo("There are ".$rst->{RecordCount}."
records in the Recordset");
```

Copyright Sammamish Software Services 2003. All rights reserved.

36

Acquiring Test Data

- Use SQL to populate tables

```
INSERT INTO CustomerTest
SELECT * FROM Customers;
```
- Using a Union:

```
SELECT (firstname + ' ' + lastname) as
name, city, postalcode into TestTable
FROM employees
UNION
SELECT companyname, city, postalcode
FROM customers
```
- Investigate automated test generation tools www.testdata.com

Copyright Sammamish Software Services 2003. All rights reserved.

37

Useful Queries for Data Verification

- Detecting duplicate data (using group by)

```
SELECT lastname+firstname,  
       Count(lastname+firstname)  
FROM Employees  
GROUP BY lastname+firstname  
HAVING Count(lastname+firstname) > 1;
```

- Finding most recent data (using a subselect)

```
SELECT * FROM orders  
WHERE orderdate =  
       (select max(orderdate) from orders)
```

Copyright Sammamish Software Services 2003. All rights reserved.

38

Concurrency and data accuracy

- Test case for concurrency
 - using SQL to start a transaction and lock a table
 - use a query that spans the entire table
update customers set row1 = 'new value'
 - observe the application's response to the locked table
- Poor response to concurrency affects data accuracy.
 - If the session hangs, what state is the data in?

Copyright Sammamish Software Services 2003. All rights reserved.

39

Standard database tests should include blocking critical tables in the database and noting the application's response.

Expected response could be to provide an informative and diplomatic message to the user and allow an option to rollback to a stable state.

A failure would be indefinite blocking and hanging of the application. In this case, the tester should observe the resulting state of the data. Was an open transaction rolled back or committed without receiving customer verification? This could cause incorrect orders to enter the system, for example.

Multiple SQL scripts can be run to simulate multiple users providing more opportunities to evaluate concurrency issues in the database.

What tools are available to support your testing efforts?

- CompuWare DevPartnerDB
- Scandiasoft DBValidator
- dbUnit SourceForge (also has many db schema comparator tools)
- These tools support some, but not all aspects of a full spectrum data access layer test plan so
 - Plan to have DBA level experienced testers on test team *writing test harnesses in SQL*

Session Summary

- Today's systems are too complex to have the luxury of testing at a single layer
- Database systems programming – accessing multiple data stores – has become an integral piece of most applications and *it is currently not being tested sufficiently in many organizations*
- Test teams need RDBMS skills now! You can achieve this by adding DB skills to your team:
 - Increase focus on hiring and training in this area
 - Be aware of existing and new tools to aid in DB testing