Database Unit Testing Fundamentals

- Eric Selje

- April 6, 2013

Eric Selje

Salty Dog SOLUTIONS

- Developer Since 1985
  - dBase/FoxBase/FoxPro since 1986
  - .Net / SQL Server more recently
- Salty Dog Solutions, LLC (Consulting)
  - Mobile
  - Web
  - Database
  - Legacy
- User Groups
  - MadFox since 1995
  - Geek Lunch

Agenda

- What is “Unit Testing”
- What does it mean to “Unit Test” a database?
- How to do it in SSMS
  - Manually coding
  - tSQLt Framework
  - SQL Test

What is “Unit Testing”

- Code that exercises a *specific portion* of your codebase in a particular context.
- Returns a simple pass/fail based on an expected known result.
- Inspect the results to know how things stand and possibly act on it.

Goals

- Catch mistakes as early in the process as possible.

Goals

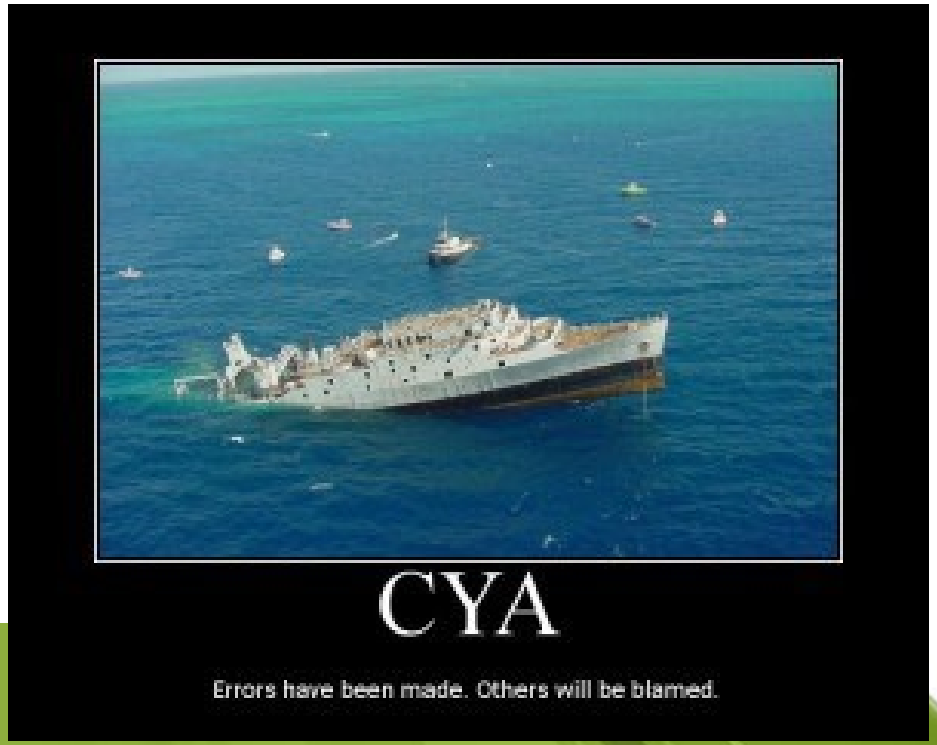
- Keep your units of code as small and testable as possible.
  - Tighter, more maintainable code

Goals

- Ensure that you have a good idea what the code is supposed to do *before* you write it.

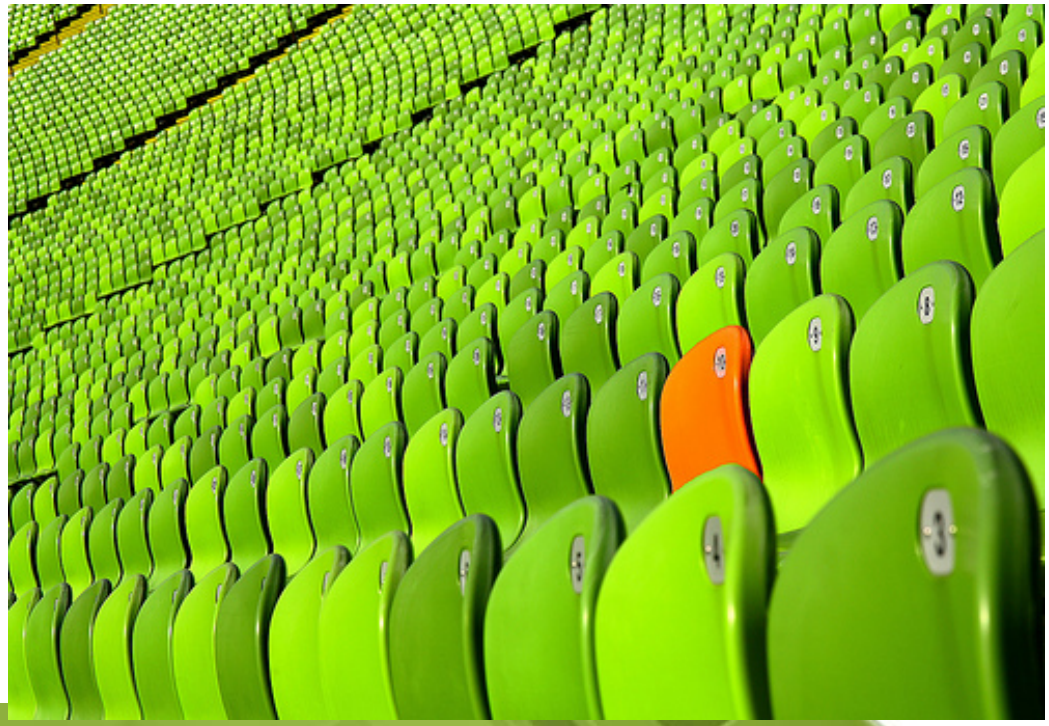
Goals

- They give you confidence that your code works as you expect it to work.



Goals (for Database folks)

Can ensure CONFORMITY with Standards



What to Test?

- Not possible to test *everything*
  - You'd like to hit 75% coverage
- Every method, possibly multiple tests
  - Normal Conditions
  - Edge Conditions
  - Null / Empty parameters
  - Out of bound conditions

Problems with Unit Testing

- More up-front time / work
  - Less instant gratification
  - Pays off on the back end
- Hard to justify to upper management
  - “Squishy”
  - Can’t prove it worked

Unit Testing is not….

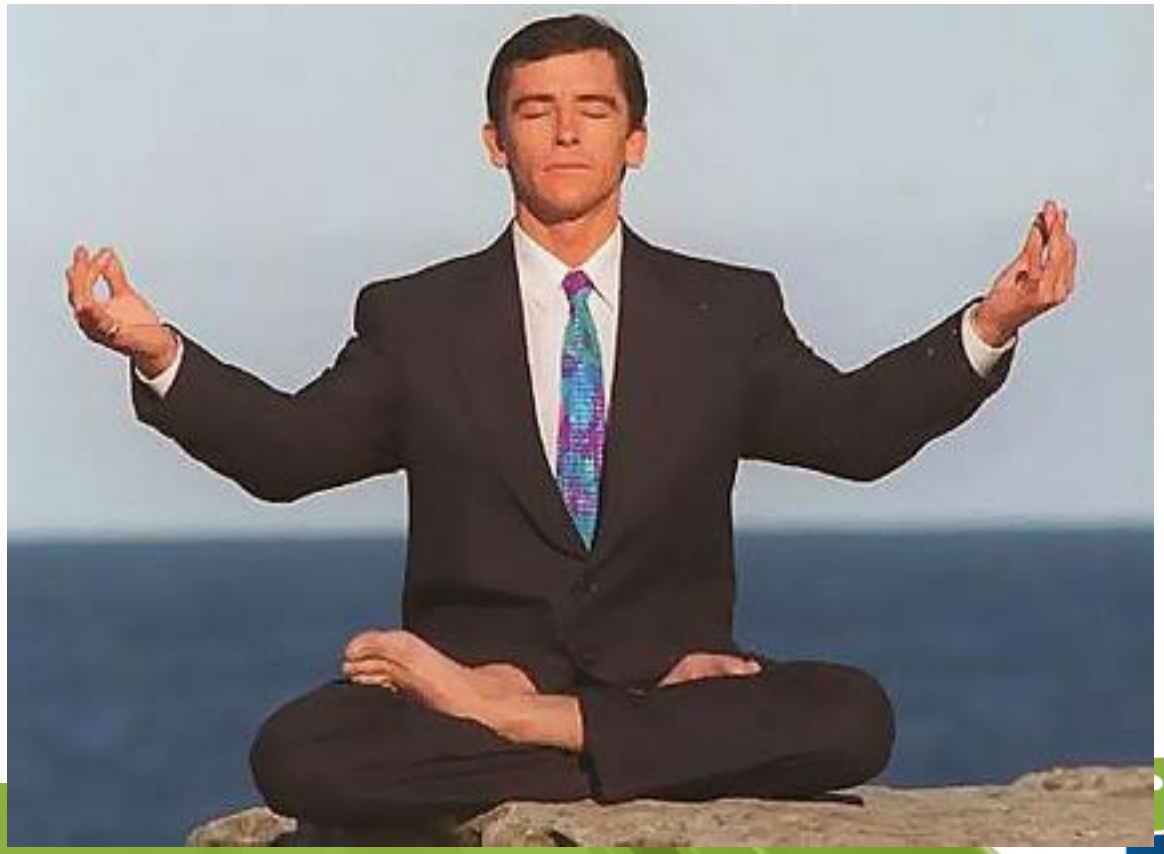
- *Integration Testing* in which we make sure our individual modules work well together.
  - Does the 'Delete' button on the U/I actually invoke the 'Delete' method in the ORM properly and handle the result?
  - Does calling the 'Delete' method in the ORM actually remove a record from the database?

Unit Testing is not….

- *System Testing*, when you deliver your app to the testers and they check it all out to make sure it meets the requirements. Eg.
  - load testing
  - compatibility testing
  - accessibility testing
  - and many more.

Unit Testing is…

- Peace of Mind



What does this have to do with *DATABASES?*

Well, there IS code in databases

- Stored procedures
- Functions
- Constraints
- Views

Example #1

- Write a unit test to ensure our scalar function returns what we think it should.
- Also, make sure it *doesn't* return what we don't think it should. Test edge cases.

Example #2

- Write multiple unit tests for a function…

Example #3

- How to test constraints?
  - Embed in a TRANSACTION
  - Try to add some bad data
  - See if it threw an error
  - End TRANSACTION

Example #4a - Views

Old way of testing

1. Alter a view
2. Run view
3. Got data? Good to go!

Example #4b - Views

How to test if a View's data went bad?

1. You'd have to have 'saved' a copy of your view data before you make changes.
2. Make your changes
3. Get your new view data
4. Compare (um…how to do that?)

This is getting out of hand!

- We *could* write the unit tests in Visual Studio
  - nUnit
  - Sql Server Data Tools
- Requires a separate tool which we may not have.
- It'd be nice if there was a framework for writing unit tests that allowed us to remain within SQL Server Management Studio.

tSQLt Framework

- http://tsqlt.org/
- tSQLt is a database unit testing framework for Microsoft SQL Server.
- Features:
  - Test are run within transactions
  - Tests are grouped within a schema
  - Output can go to a file that a CI tool can use
  - …and more

tSQLt Framework Installation

1. Download zip
2. Add scripts to a project (optional)

Comes w/ an example database…

Adding tSql to your database

1. Prep your database
    1. ALTER AUTHORIZATION ON DATABASE TO sa (Maybe)
    2. SET TRUSTWORTHY ON
2. Run tSqlt.class script

    on each database to test

Redo our tests in tSQLt

- Create a test class: tSQLt.NewTestClass
  - Creates a new schema with that classname
- Create a test: CREATE PROCEDURE
  - Schema should be the new class name
  - Prefix the sp name with ‘test’
- Run the Test: tsqlt.Run <testName>

Assert Yourself

- AssertEquals
- AssertLike
  - % or _
- AssertEqualsString
  - Varchar(Max) comparisons

Assert Yourself

- AssertEqualsTable
- AssertResultSetsHaveSameMetaData

- FakeTable (remove constraints)

Assert Yourself

AssertObjectExists

Run All Tests in a Class

- One test per stored procedure
- tsqlt.RunTestClass <className>

- tsqlt.RunAll

Faking It

Tips

- *Filter the Stored Procedures to just see Tests*
- *Assign a hotkey to run tests (Tools, Options, Environment, Keyboard, EXEC tSQLt.RunAll; --)*
- *Create a template for creating a test (thanks, @SQLAgentMan)*

Not Enough?

- *No Pretty Color Coding?*
- *I can't just doubleclick on my failed test to edit it? Waah?*

SQL Test

redgate
ingeniously simple tools

- RedGate
- Not Free
  - $295 Standalone
  - $1495 in SQL Developer Bundle
    - SQL Source Control
    - Continuous Integration
  - Except for one lucky attendee at #SQLSat206

SQL Test

redgate
ingeniously simple tools

Gets you

- Nice GUI
- Color Coding
- SQL Cop Tests

Demo…

Continuous Integration

- *Automatically* run Unit Tests
  - On a schedule
  - When code is checked in
    - Reject if fails
    - Proceed with other tests if succeeds
    - Compile the app
    - Distribute

Conclusion

- Unit Testing *is* for Database People
- Discipline you to write testable code
- Reward you with easy to maintain code
  - Fewer errors
  - Save money
  - Save time
- Conform to standards

Q&A / Comments

- What else would you consider testing?

Contact Me

Twitter: EricSelje
LinkedIn: saltydog

Resources

- Unit Testing Database with tSQLt
- Ten Things I Wish I'd Known
- 48 SQL Cop Tests (Zip File)
- Using SQL Test in Continuous Integration, by Dave Green
- How to write good unit tests
- Are Unit Tests overused?
- Test Driven Development