Testing! We're Talking About (Unit Testing)... Your Database

Dan Mallott
Head of Platform & Infrastructure
@ e-conomic



e·conomic

Who Am I?

Dan Mallott
Head of Platform & Infrastructure
@ e-conomic

- Software Developer (and occasional DBA) since 2011
- Primary experience with Microsoft tech
- I referee and play ice hockey

~~Twitter~~ X: @danielmallott

Github: github.com/danielmallott

LinkedIn: linkedin.com/in/danielmallott



e·conomic

What's On Deck

- Introduction
- Unit Testing Basics
- Manual Test Setup
- tSQLt
- CI Integration

```
-- Arrange
EXEC tSQLt.FakeTable @T
DECLARE @newUserId INT;

-- Act
EXEC @newUserId = dbo.I

-- Assert
DECLARE @countInTable I
EXEC tSQLt.AssertEquals
EXEC tSQLt.AssertEquals
```

e·conomic

Introduction

e·conomic



Unit Testing, also known as Component Testing, is a level of software testing where individual units/components of a software are tested

e·conomic

What’s a Unit?

e·conomic



A unit is the smallest piece of code that can be logically isolated in a system

e·conomic

What is the Purpose of Unit Testing?

Benefits

- Find logical issues early
- Creates a code contract that protects against changes
- Ensures adherence to acceptance criteria
- Provides living documentation of how the code is meant to be consumed

Drawbacks

- Does not catch integration issues
- Can be difficult to set up realistic unit tests
- Adds extra development effort – sometimes more than twice as much as the initial code
- Requires tests to be run to be effective

e·conomic

Characteristics of Good Unit Tests

→ **Automated**

Should be able to be run automatically

→ **Granular**

Should only test one branch at a time

→ **Fast**

Should execute very quickly

→ **Isolated**

Are testing only the code under test

→ **Deterministic**

Produce the same result every time

→ **Independent**

Are not reliant on execution order

e·conomic



Should we be unit testing database code?

e·conomic



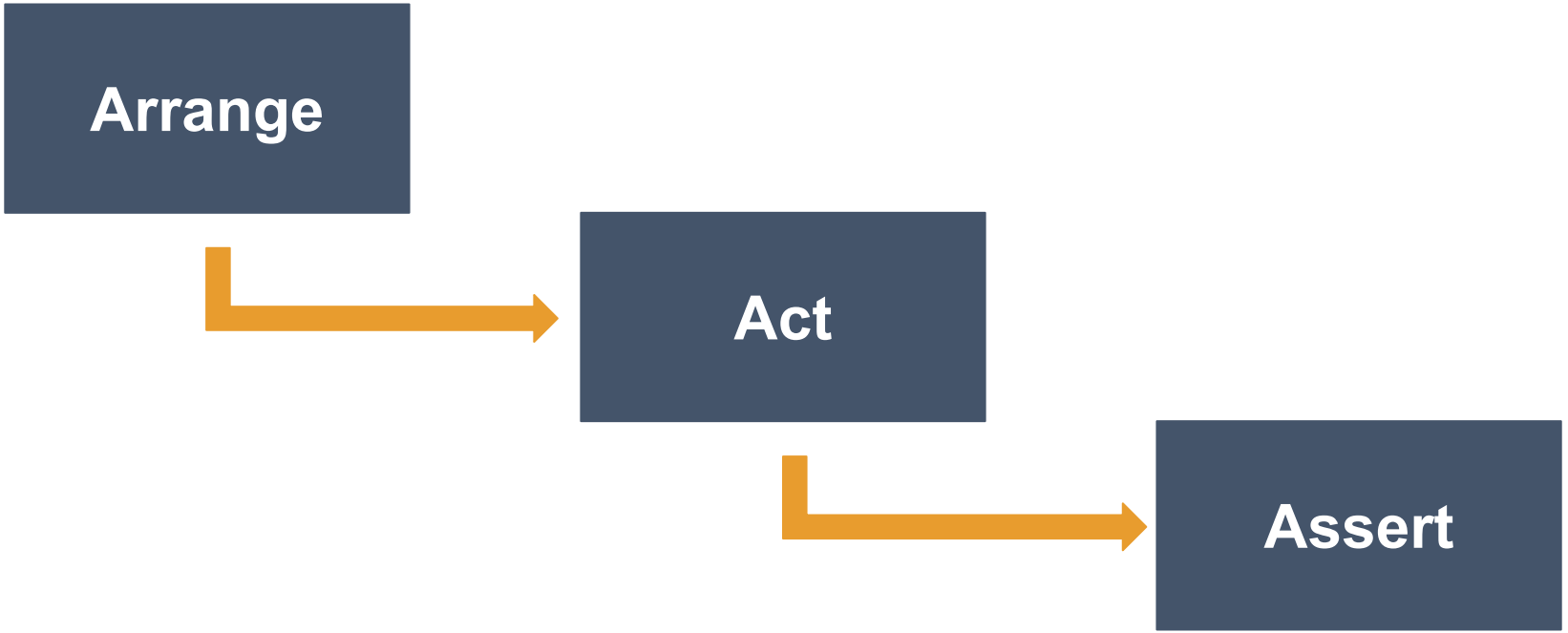
The better question is: How good are your integration tests?

e·conomic

Unit Testing Basics

e·conomic

The 3 A's of Unit Testing



e·conomic

The Arrange Step



e·conomic

The Arrange Step

Set up preconditions

Create all the conditions necessary for the test - including any processes or procedures that need to happen before the code under test is invoked

Create testing data

Create the data that will be the input for your test

Also create a model of the expected output (if any)

Mock dependencies

Mock any dependent code to return data or status codes as you expect during the test - ensure only the code under test is invoked

e·conomic

The Act Step



e·conomic

The Act Step

Invoke the Code

Actually invoke the code under test, using the data set up in the Arrange step

Record Results

If applicable, save the results of the code to a variable, or otherwise capture the results for later comparison

e·conomic

The Assert Step

Check Your Results

Take the results of the Act step and compare to what you expected to get.

Matches indicate success; otherwise, the test has failed



e·conomic

Manual Test Setup

e·conomic

Running Unit Tests Manually

With a Dash of Powershell

Writing tests manually is entirely possible - we all do it anyway!

Adding a dash of Powershell to the mix can help us run all of them

e·conomic

DEMO

e·conomic



That’s kind of a pain…

e·conomic

tSQLt

e·conomic

What’s a “tSQLt”?

Meet the SQL Server Unit Testing framework

tSQLt allows us to automate lots of the “Arrange” and “Assert” steps by giving us a Mocking and Test Execution framework

It is also opinionated, so your code has to be written a “certain” way to take full advantage of it

e·conomic

Mocking? Not Just for Making Fun of Oracle...

Say you are inserting to a table...

Maybe you don't want to insert to the actual table

Maybe the table is not important to the logic you are testing (remember - it is a dependency of the procedure)

We can mock (or fake) the table in the ***Arrange*** step

```
-- Arrange
EXEC tSQLt.FakeTable @TableName = 'dbo.Users', @Identity = 1;
DECLARE @newUserId INT;

-- Act
EXEC @newUserId = dbo.InsertUser 'Test', 'Test', 'Test', 'Test';

-- Assert
DECLARE @countInTable INT = (SELECT COUNT(1) FROM dbo.Users);
EXEC tSQLt.AssertEquals 1, @countInTable, 'Did not insert row';
EXEC tSQLt.AssertEquals 1, @newUserId, 'Did not insert new Id';
```

e·conomic

A Different Way of Writing T-SQL

Componentized

You will prefer small, reusable chunks of database logic

Common Language Features

Depending on the testing framework, some things might be harder to test, so you will avoid them

Testability

You will prefer testability over optimization, until you need to optimize

e·conomic



Begone, 5000-line stored procedures!

e·conomic

DEMO

e·conomic

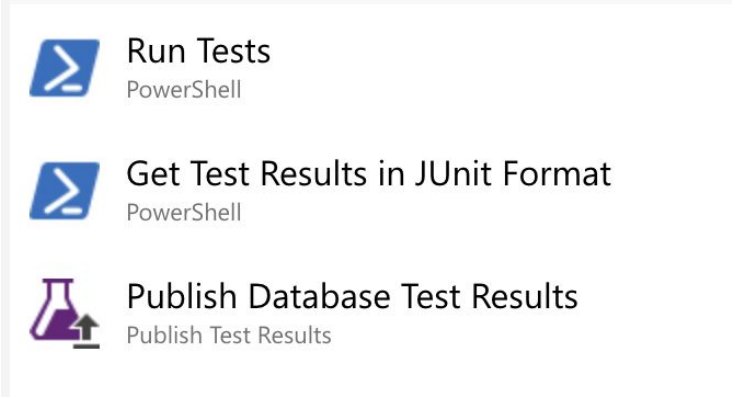
CI Integration

e·conomic

Adding This to a Build Pipeline

Thanks to cloud services, containers, and Linux compatibility in the SQL Server world, it is much easier today

Some Powershell and the build tool you already have will allow you to run "CI" on your database code, including the Unit Tests



Script *

```
Invoke-Sqlcmd -ServerInstance "test-referee-scheduling-system-database.eastus.azurecontainer.io" -Database
"RefereeSchedulingSystem" -Username "sa"  -Password [redacted] -Query "EXEC tSQLt.RunAll" -ConnectionTimeout 120
```

Script *

```
$results = Invoke-Sqlcmd -ServerInstance "test-referee-scheduling-system-database.eastus.azurecontainer.io" -Database
"RefereeSchedulingSystem" -Username "sa"  -Password '[redacted]-Query "EXEC tSQLt.XmlResultFormatter" -
ConnectionTimeout 120
Write-Host "Test results:"
Write-Host "$results[0]"
$results[0] > $(System.DefaultWorkingDirectory)\tSQLt_testResults.xml
```

Test result format *

JUnit

Test results files *

tSQLt_testResults.xml

Search folder

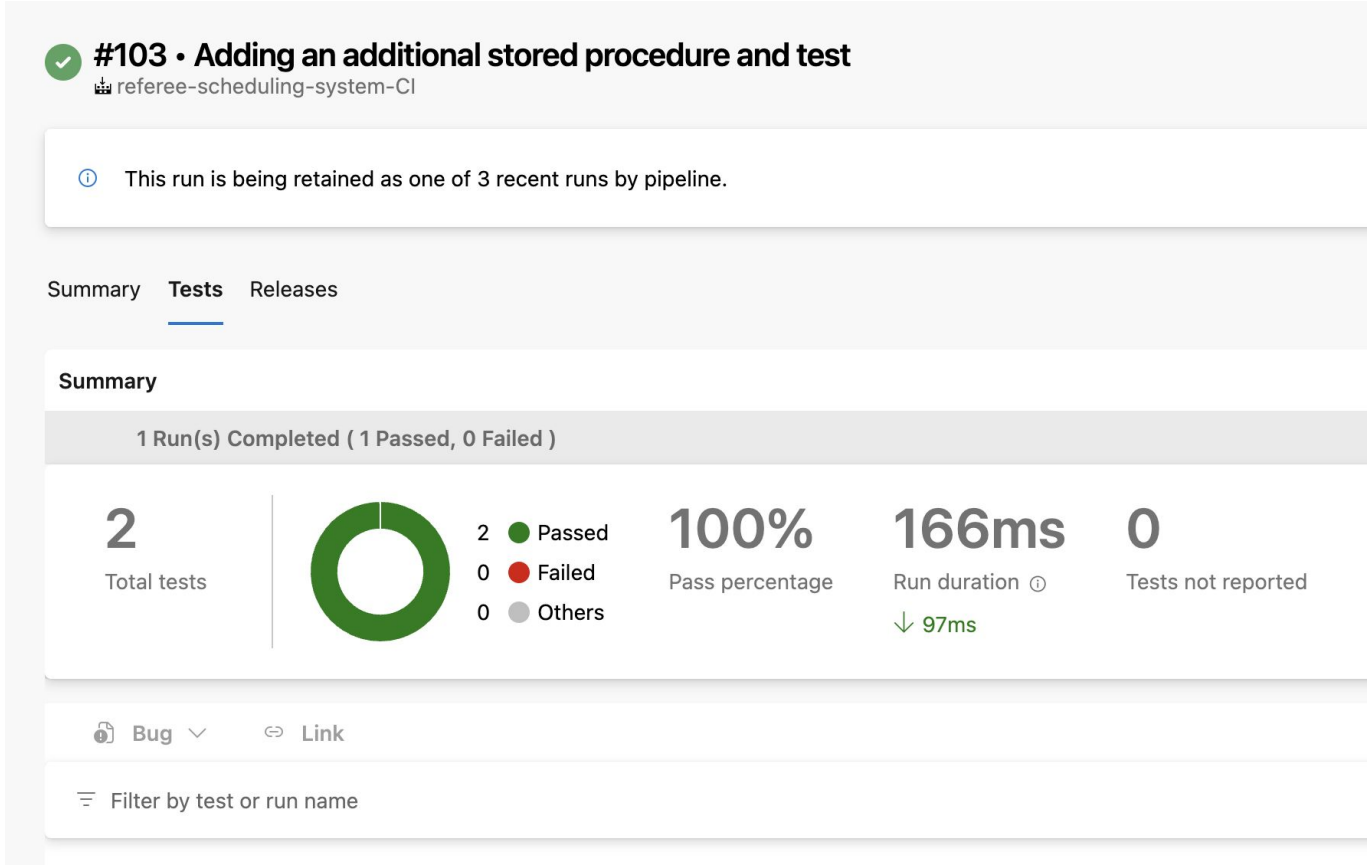
$(System.DefaultWorkingDirectory)

Merge test results

Fail if there are test failures

e·conomic

And You Can Get a Nice Report



e·conomic

Questions?

e·conomic

Your Feedback is Needed!



https://sqlkonferenzeval.azurewebsites.net/129455178333461

e·conomic

Thank You!

~~Twitter~~ X: @danielmallott

Github: github.com/danielmallott

LinkedIn: linkedin.com/in/danielmallott

e·conomic

Flexibility
for the future

e·conomic