Unit Testing T-SQL code with tSQLt

Isolating Dependencies

Dave Green
Twitter: @d_a_green
Email: dave@dgta.co.uk

pluralsight
hardcore developer training

Why Isolate

- **Prevent test behavior being changed except by what is under test**
  - Data
  - Table constraints
  - Other Stored Procedures which are called
  - Functions

Isolating data and tables

- **Enough test data to do the test**
  - We are testing functionality, not performance
  - Unit tests need to run quickly, so they can be run frequently
  - Realistic data should be used – in small amounts

- **Delete data / objects in the system and use rollback**
  - Potential issues with foreign keys, constraints
  - Could take a while
  - Hard to write without introducing complications
  - *Doesn't isolate all dependencies*

There must be a better way..

Isolating data and tables

- tSQLt.Faketable
- Takes a parameter @TableName to denote object to fake
- Moves object and creates a copy
    - Without constraints
    - Without data
- Code under test remains unchanged
- Minimizes setup work required
- Test is repeatable

Duplication of test setup

- Setup Routines minimize duplication
- Named SetUp and created in the Test Class (schema)
- Run before each test
- Rolled back automatically after the test

Isolating from other Stored Procedures

- **tSQLt.SpyProcedure**
  - @ProcedureName denotes the stored procedure to isolate from
  - @CommandToExecute will be executed in place of the stored procedure
- **Table created to log when isolated procedure is called**
  - *@ProcedureName*_SpyProcedureLog
  - Records the parameters passed to isolated procedure
- **Remember to unit test the isolated procedure too!**

Report contacts and average duration

- **For Each Contact Type**
    - How many contacts have taken place
    - How much time in total was used?

- **For interactions that started within X complete months before a specified date**

- **Example output:**

InteractionTypeText	Occurrences	TotalTimeMins
Meeting	150	50000
Introduction	200	20450
Phone Call	230	34572

Isolating Functions

- **No built in mocking structure**
- **tSQLt.RemoveObject**
  - Removes the object
  - Also useful to remove objects where they exist but you want to check a routine will create them
- **You can then implement a stub**
- **No logging that the stub was called**

Limitations of isolation

- **Code that relies on the current date/time**
  - Calculate test data dynamically so that the relative scenario remains
  - As time moves on “the last three days” means different dates

- **Objects constrained by Schemabinding**
  - SQL server will correctly prevent mocking
  - If schemabinding is required, you must test without mocking

Summary

- We can isolate our code from objects on which our code under test depends, but which are not themselves under test
- tSQLt provides easy mechanisms to isolate tables, views and utility procedures
- You can also move objects out of the way and create stubs manually if you want.
- Test data should be chosen with care to ensure that your test continues to be repeatable as time passes.