sql TEST

tSQLt QUICK REFERENCE GUIDE

redgate

tSQLt Framework

tSQLt is the open source database unit testing framework used in SQL Test. tSQLt is compatible with SQL Server 2005 (service pack 2 required) and above on all editions.

tSQLt allows you to implement unit tests in T-SQL as stored procedures. The framework itself is a collection of objects that instal into a schema named tSQLt. The following tSQLt features make it easier to create and manage unit tests.

Key Points

Tests are automatically run within transactions – this keeps tests independent and removes the need for clean-up code, as all changes are automatically rolled back.

Tests can be grouped together within a schema – allowing you to organize your tests and use common setup methods.

Results can be output as text or JUnit XML – making it easier to integrate with a continuous integration tool.

Mocking features to fake tables and views, and create stored procedure spies – allowing you to isolate the code which you are testing.

Assertions

AssertEquals

Compares two values for equality.

```
tSQLt.AssertEquals [@Expected = ] expected value
            , [@Actual = ] actual value
            [, [@Message = ] 'message' ]
```

AssertEqualsString

Compares two string values for equality.

```
tSQLt.AssertEqualsString [@Expected = ] expected value
            , [@Actual = ] actual value
            [, [@Message = ] 'message' ]
```

AssertEqualsTable

Compares the contents of two tables for equality.

```
tSQLt.AssertEqualsTable [@Expected = ] 'expected table name'
            , [@Actual = ] 'actual table name'
            [, [@FailMsg = ] 'message' ]
```

AssertObjectExists

Checks to see if an object with the specified name exists in the database.

```
tSQLt.AssertObjectExists [@ObjectName = ] 'object name'
            [, [@Message = ] 'message' ]
```

AssertResultSetsHaveSameMetaData

Compares the meta data (column names and properties) of results for two commands.

```
tSQLt.AssertResultSetsHaveSameMetaData
[@expectedCommand = ] 'expected command'
```

Fail

Simply fails a test case with the specified failure message.

```
tSQLt.Fail [ [@Message0 = ] message part ]
```

Isolating Dependencies

FakeTable

Replaces a table with a fake table, without data and constraints.

```
tSQLt.FakeTable [@TableName = ] 'table name'
            , [[@SchemaName = ] 'schema name']
            , [[@Identity = ] 'preserve identity']
            , [[@ComputedColumns = ] 'preserve computed columns']
            , [[@Defaults = ] 'preserve default constraints']
```

ApplyConstraint

Adds back constraints to a faked table so they can be tested independently.

```
tSQLt.ApplyConstraint [@TableName = ] 'table name'
            , [@ConstraintName = ] 'constraint name'
            , [@SchemaName = ] 'schema name'
```

SpyProcedure

Replaces stored procedure functionality with logging.

```
tSQLt.SpyProcedure [@ProcedureName = ] 'procedure name'
            [, [@CommandToExecute = ] 'command' ]
```

AdventureWorks Example Test

```
CREATE PROCEDURE [MyNewTestClass].[test update employee]
AS
BEGIN
EXEC tSQLt.FakeTable 'HumanResources.Employee';
INSERT INTO HumanResources.Employee (EmployeeID, Gender)
VALUES (0, 'M');

EXEC HumanResources.uspUpdateEmployeePersonalInfo
                    @EmployeeID = 0,
                    @NationalIDNumber = NULL,
                    @BirthDate = NULL,
                    @MaritalStatus = NULL,
                    @Gender = 'F';

DECLARE @ActualGender CHAR(1);
SET @ActualGender = (SELECT Gender FROM HumanResources.Employee);

EXEC tSQLt.AssertEquals @Expected = 'F', @Actual = @ActualGender;
END;
```