ExecuteTestSql PowerShell Script

# Overview

With EDennis.NetCoreTestingUtilities and EDennis.MigrationsExtensions, you can store test parameters and expected JSON output in a \_maintenance.TestJson table. The expected JSON can be generated in any way; however, one convenient approach is to use SQL Server queries with the FOR JSON clause and to save the entire test record with the EDennis \_maintenance.SaveTestJson stored procedure. Using this approach, you can create .SQL files (in your test project) that generate the test parameters and expected JSON.

Unfortunately, if you drop and re-create your database, the TestJson records will be lost. Also, if you are working in a team environment, where other team members create TestJson .SQL files, it may be burdensome to find and manually execute these files on different developer PCs. To address these issues, I developed a PowerShell script, "ExecuteTestSql.ps1," that will re-create all of your TestJson records by executing all .SQL files in projects having the word “Test” in their project directory names. The PowerShell script appears below. Note that it is designed to work with LocalDb, but you can change the script to point to any SQL Server database.

To make ExecuteTestSql readily available from the Package Manager prompt, perform the following tasks:

1.       At the Package Manager prompt, create a new profile file

**PM> if (!(Test-Path $Profile)) { New-Item -Type file -Path $Profile -Force }**

2.       At the Package Manger prompt, launch the new profile file in PowerShell ISE

**PM> PowerShell\_ISE.exe $Profile**

3.       Paste in the code below and save the file

4.       Restart your computer

5.       From this point on, the following code should work without any additional work:

PM> ExecuteTestSql -database "MyDatabase" -showSuccesses $false

# #######################################################

#

# Author: Dennis Mitchell

# Last Modified: 2018-12-24

# Functionality: Executes all .SQL files in test projects

#

# #######################################################

function ExecuteTestSql($database,$showSuccesses){

    #declare a dictionary for holding the SQL strings

    $dict = new-object "System.Collections.Generic.Dictionary``2[System.String,System.String]";

    $cntPrev = 0;

    $cxn = new-object system.data.sqlclient.sqlconnection("Server=(LocalDB)\MSSQLLocalDB;Database=$database;Trusted\_Connection=True;MultipleActiveResultSets=True;");

    $cxn.Open();

    #purge the TestJson table

    $cmdTruncate = $cxn.CreateCommand();

    $cmdTruncate.CommandText = "exec \_maintenance.TruncateTestJson";

    $cmdTruncate.ExecuteNonQuery() | Out-Null;

    #define SQL query for counting all records in TestJson

    $sqlCount =

@'

    select sum(cnt)

        from (

            select count(\*) cnt

                from \_maintenance.TestJson

            union select count(\*) cnt

                from \_maintenance.TestJsonHistory

         ) a;

'@

    #define SQL query for getting the most recently added key

    $sqlKey =

@'

    select top 1 ProjectName

            + '|' + ClassName + '|' + MethodName

            + '|' + TestScenario + '|' + TestCase

            + '|' + TestFile KeyVal

        from \_maintenance.TestJson

        order by SysStart desc;

'@

    # get all .sql files in projects having the word "Test" in them

    Get-ChildItem -Path \*.sql -Recurse -File |

        Where-Object {

            $\_.PSParentPath -like "\*Test\*" -and $\_.FullName `

                -notmatch "\\bin\\" -and !$\_.PSISContainer

        } |

        Select FullName |

        ForEach-Object {

            $file = $\_.FullName;

            try {

                #execute the SQL file

                $sqlFile = [IO.File]::ReadAllText($file) ;

                $cmdFile = new-object system.data.sqlclient.sqlcommand($sqlFile, $cxn);

                $cmdFile.ExecuteNonQuery() | Out-Null;

                #get the total number of records in TestJson and history table

                $cmdCount = new-object system.data.sqlclient.sqlcommand($sqlCount, $cxn);

                $cnt = [int]$cmdCount.ExecuteScalar();

                #if the count increased, then ...

                if($cnt -gt $cntPrev){

                    #get the key for the last inserted record

                    $cmdKey = new-object system.data.sqlclient.sqlcommand($sqlKey, $cxn);

                    $keyVal = [string]$cmdKey.ExecuteScalar();

                    #if the key already exists, append the additional

                    #  file name to the existing file name

                    if ($dict.ContainsKey($keyVal)){

                        $dict[$keyVal] = $dict[$keyVal] + "`n`t" + $($file);

                    #otherwise, just add the new file name

                    } else {

                        $dict.Add($keyVal,$file);

                    }

                    if($showSuccesses){

                        Write-Host "    Successfully executed:  $($file)";

                    }

                }

                $cntPrev = $cnt;

            } catch {

                #write the error message to the console

                Write-Host "\*\*\*\*ERROR: COULD NOT EXECUTE:  $($file): $($\_.Exception.Message)";

            }

        } | Out-Null;

    #write all duplicate test parameters to the console

    foreach($key in $dict.Keys){

        if(([string]$dict[$key]).Contains("`n`t")) {

            Write-Host "Duplicate Test Params: $([string]$key) : `n`t$([string]$dict[$key])";

        }

    }

}