**Database Unit testing guide**

Unfollow

2

Edit

Michael Mateta

13 Jan

**Unit Testing with SQL Database project**

In this example we will go through how we can create and run unit tests on a code change, this is worked through an example being implemented. Here I have created a procedure with the below logic, for example: proc ssd.spMerge\_projectAnswer which merges the data from Staging table: tblProjectAnswer to ssd final table: tblProjectAnswer

CREATE PROCEDURE [ssd].[spMerge\_projectAnswer]  
AS  
BEGIN  
MERGE INTO [ssd].[tblProjectAnswer] AS t  
USING [Staging].[ssd\_tblProjectAnswer] AS s  
ON s.AnswerText = t.AnswerText  
WHEN NOT MATCHED BY TARGET THEN  
INSERT([AnswerText]) VALUES (s.[AnswerText])  
END

Firstly I create the procedure in my database project, once done and happy with the logic, we then right click on the project's object file: spMerge\_projectAnswer.sql, then click create unit test as shown below:

This will then lead you to the below page where you can label the unit test you are creating, for the purposes of standardization in this repo, we can name unit tests using the below naming convention:  
Schema\_ObjectNameUnitTest.cs in this example I have named it: ssd\_spMerge\_projectAnswerUnitTest.cs

After that this will then lead you to the below screen which is where we start actually testing:

Next step is renaming the unit test, we would want to rename the test as below, so Click on the Rename button which will give you a pop-up. This name should be that of what the test is actually doing and you can have multiple different tests in one unit tests that do different things to the same code. For this test we want to Insert data and check the output data matches what we expect when merged:

Then after that we click on the *Click here to create*, this will the create the UDI for the test.

In order to set up the conditions for the unit test, we would need to click on pre-test in the drop down below, We can include pre and post test SQL statements which will be run before and after the test script is executed, respectively.  
For example, we would like to have a clean [ssd].[tblProjectAnswer] and [Staging].[ssd\_tblProjectAnswer] table before the unit test runs, we can have the following SQL script to truncate all rows in the table. Bearing in mind the [ssd] tables have table key relationships persisted throughout thus we need to manage those constraints as part of this script.

**WARNING!**

Before you run any unit tests, make sure you have your connection set up to the correct database.  
SERVERNAME: [sqlsvr-warehouse-devci-001.database.windows.net](http://sqlsvr-warehouse-devci-001.database.windows.net/)   
DATABASE: *sqldb-survey-devci-001*  
USER: *dba\_unittest*  
You can acquire the password for this user from the Data Engineering team personnel or secrets location: *Warehouse-Login-dbaUnitTest-Password*

This is to avoid any possible truncation or breaking code within other environments such as dev or devtest. This database can also be used for testing your scripts for the unit testing within management studio.

In this pre-test, we will insert sample data both tables to run the merge and check if it correctly updates the ssd table:  
ALTER TABLE [ssd].[tblProjectScaleAnswer] DROP CONSTRAINT IF EXISTS [FK\_ssd\_tblProjectScaleAnswer\_AnswerID]

TRUNCATE TABLE ssd.[tblProjectAnswer]

TRUNCATE TABLE [Staging].[ssd\_tblProjectAnswer]

INSERT INTO ssd.[tblProjectAnswer]([AnswerText]) VALUES ('Answer 1');

INSERT INTO [Staging].[ssd\_tblProjectAnswer] VALUES ('Answer 1');  
INSERT INTO [Staging].[ssd\_tblProjectAnswer] VALUES ('Answer 14');  
INSERT INTO [Staging].[ssd\_tblProjectAnswer] VALUES ('Answer 5');

From the above data sample, my test is as follows, I have text: “Answer 1” already existing in the tblProjectAnswer, I then hae 3 different answer texts in the staging table including “Answer 1”. However because “Answer 1” already exists in the ssd so I expect the merge to check it exists and only insert the new answer texts from the staging which do not exist, thus only 2 new rows are added to ssd.tblProjectAnswer.  
We now have the below pre-test:

In order to run the test, I have the code as follows:  
SELECT AnswerID, AnswerText FROM [ssd].[tblProjectAnswer]

EXECUTE [ssd].[spMerge\_projectAnswer]

SELECT AnswerID, AnswerText FROM [ssd].[tblProjectAnswer]

Specifying the columns will help greatly when running unit tests incase of table changes which can affect the schema of the table without breaking your unit test. I will then also have the below row count tests for resultset1 and 2. The resultset refer to the select statements being returned in the query. These are the test conditions for my test, you can have as many test conditions as you see fit but keeping them small and manageable will help when running and reviewing multiple unit tests. See below screen shot

We change the properties of the test conditions in the properties tab.  
Once this is done it is important to clean up the tables in your post test scripting as any data left in the table can potentially break your unit tests. Make sure to re-add the FK relationships or any constraints which had been removed previously.

Now we are ready to run the unit test. First we must publish our changes to the database, so go to the publish script under survey in the db project: sqldb-survey-devci-001.publish.xml(*Note* for the first publish you may need to use the connection details above when publishing to the database, avoid dev or devtest for publishing code from the database project as that should go through DevOps via the normal Pull Request).  
Double click the script to get the following popup:

Click Publish, this is to release your db project changes to the devci server which is dedicated just to unit testing. Once the publish is completed as below we are now ready to run the unit tests. Make sure to save your work as you go by clicking Save All to save the entire project.

Now go to Test in your Menu bar, click on Test Explorer. Test Explorer is where we run the database tests.

Click on Run all tests to run the new unit test you created along with every other test in the project for Continuous Integration:

Our tests ran successfully as shown below. Now we can Save All again and we can deploy the project to DevOps using the usual process.

0 visits in last 30 days