Table of Contents

[Functions Used 2](#_Toc428261012)

[Mocking of production table 2](#_Toc428261013)

[Using SqlExecuter 3](#_Toc428261014)

[Test Scenarios 3](#_Toc428261015)

[General Suggestions 3](#_Toc428261016)

[References 3](#_Toc428261017)

# Functions Used

|  |  |
| --- | --- |
| **Functions** | **Description** |
| Suites | A test suite is defined with the keyword “*describe*” with two parameters, the first is a String which is a title of test suite and second is function which is a block of code that implements the suite. |
| Specs | A spec is defined with the keyword “*it*” with two parameters, the first is a String which is a title of a spec and second is function which is a block of code that implements the spec. |
| beforeOnce | This function is called once before all specs in the describe block.  *Example:* If a table needs to be cleaned before inserting new data in each spec, then instead of clearing the table in each spec, we can clear the table in beforeEach (). Therefore, the table will be cleaned before the execution of each spec. |
| beforeEach | This function is called once before each spec in the describe block.  *Example:* If a table needs to be created, then it is recommended to create it in beforeOnce (), so that before the execution of any spec, we have a table created. |
| Spies: and.callFake | When “*spy*” with “*and.callFake*” is used, then all calls to the spy will delegate to the supplied function. |
| Matchers | * toEqual * not.toBeNull * not.toEqual * toThrowError * toMatch * toBe |

# Mocking of production table

It is recommended to mock the tables while writing the test cases so that the production database tables doesn’t get affected. To achieve this, the spyOn() function provided by jasmine is used.

To achieve the mocking of tables, the package named as “TableUtils” is used which is responsible to copy the structure of a production table into mock tables in user’s schema.

Mock tables are the copy of production tables. Mocking is used for the conditions where any spec tends to make any changes in the table used. For this, mocking helps to make changes in dummy tables instead to modifying the production tables.

# Using SqlExecuter

If a function under test contains a statement which modifies the data in the mock tables, then with the use of SqlExecuter, we can query the modified table to check whether the changes has been done or not.

# Test Scenarios

* Test end-to-end functionality of a service.
* Test functions defined in service with all possible positive and negative scenarios. Test the scenarios which can lead to exceptions.

# General Suggestions

* The test suite and spec description should be defined properly in order to explain the purpose of the test.
* Insert meaningful and readable input test data.
* If a test can be performed with one set of input data, then second set is not required.
* If the method to be tested can handle multiple sets of input data, at least one test case shall test multiple sets of input data
* Test only one exception in one spec.
* Goal of writing XS Unit tests is not only to reach a as high as possible code coverage, but also to check if all the supported and not supported use cases of the productive functions are tested properly.
  + Therefore it is essential to read and understand the productive code completely and then to think of good test cases.
* Instead of hard coding the validations, do a generic loop over the result and check for each iteration.
* Multiple test cases with same expected result can be tested in one expect block using AND operator.

# References

<https://wiki.wdf.sap.corp/wiki/display/ASE/Unit+Testing+on+HANA+in+XSUnit>

<http://help.sap.com/hana/SAP_HANA_XS_Unit_JavaScript_API_Reference_en/index.html>

[https://wiki.wdf.sap.corp/wiki/download/attachments/1588440504/HANAXS\_cheatsheet\_2.0.pdf](https://wiki.wdf.sap.corp/wiki/download/attachments/1588440504/HANAXS_cheatsheet_2.0.pdf?version=2&modificationDate=1417187578000&api=v2)