

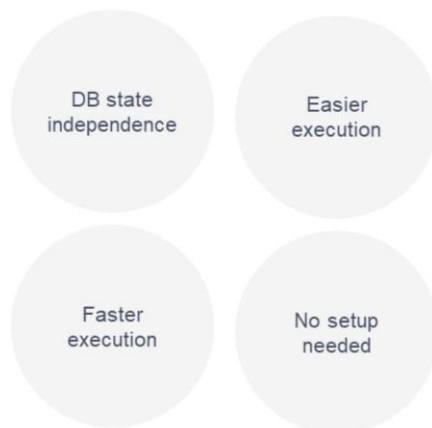
GeneXus[™]
by **Globant**

Database Mocking

GeneXus[™]
Cloud

Benefits of Database Mocking

GeneXus[™]
by Globant



Database Mocking enables you to save the data sets required for a unit test to use them in every execution of it, regardless of database state.

Mocking allows you DB state independence because regardless of the data you have in the DB, tests execution goes against the mock file saved for the executing test.

When tests execute using mocking, no previous set up is needed, since all the data required for the test in order to run is already stored in its mock file. A mocked test will have fastest execution times since no access to an actual database is performed, data is obtained directly from in memory data (loaded from its mock file).

The alternative to DB mock is to run a DB initialization script/procedure before executing the test, which is slower and hence more expensive.

Database Mocking in GeneXus

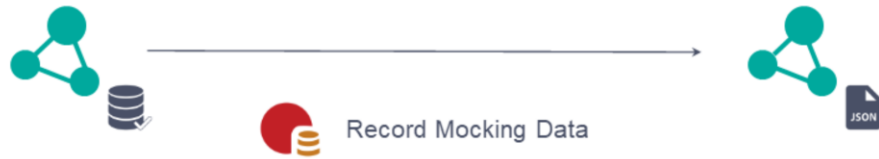
GeneXus[™]
by Globant



Database Mocking is a technique available for unit test objects, allowing to capture the desired database state (for different tables) in your tests to let specific data sets ready for future unit test execution.

So, the first step is to have in place all the data required for the unit tests to execute properly. This is, if the test requires that a register doesn't exist, or that an account has a balance of certain amount, the database must have all the required information in order to execute this test at recording time. All the setup you should do every time before executing the test if you did not have DB mocking feature

Using this technique, you can focus on getting the unit test data sets ready once (in a real database), and then use it on different unit test phases. In other words, Database Mocking is a simulation of a database with fewer records.



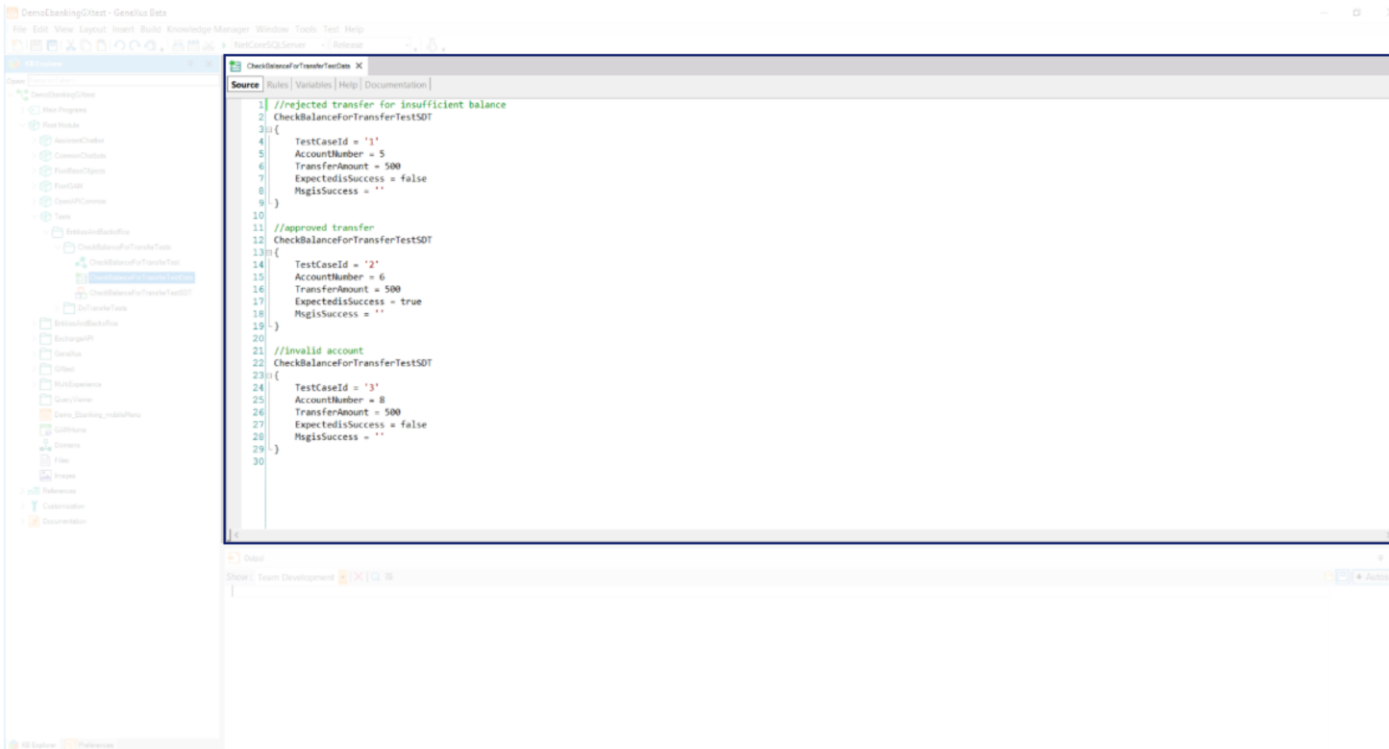
Mocking works by recording data (SQLs / results) used for a test by listening for all sentences and responses during the **Record Mocking Data operation**. In GeneXus, database sentences are stored in a json file (as **mocking data**). You will be able to save the set of data used in each unit test and share it with all the team.



Finally, once the Record Mocking Data operation ends, mocking data will be stored in a KB file object. A reference to this file is added through a property called Mock Data File in the unit test object.

Also, the Mock Data property is set to True. This allows to enable/disable the mock data usage without having to delete mock file reference.

First Database Mocking example



Let's see how to generate the database mock file for CheckBalanceForTransferUnitTest.

Remember that we tested this proc with three test cases: rejected transfer for not enough balance, an approved transfer and an invalid account.

The screenshot displays the GeneXus IDE interface. The left sidebar shows a project tree with 'DemoEbankingGXtest' selected. The main editor shows the source code for 'CheckBalanceForTransferTestSDT' with three test cases:

```

1 //rejected transfer for insufficient balance
2 CheckBalanceForTransferTestSDT
3 {
4   TestCaseId = '1'
5   AccountNumber = 5
6   TransferAmount = 500
7   ExpectedIsSuccess = false
8   MgisSuccess = ''
9 }
10
11 //approved transfer
12 CheckBalanceForTransferTestSDT
13 {
14   TestCaseId = '2'
15   AccountNumber = 6
16   TransferAmount = 500
17   ExpectedIsSuccess = true
18   MgisSuccess = ''
19 }
20
21 //invalid account
22 CheckBalanceForTransferTestSDT
23 {
24   TestCaseId = '3'
25   AccountNumber = 8
26   TransferAmount = 500
27   ExpectedIsSuccess = false
28   MgisSuccess = ''
29 }
30

```

The right pane shows a SQL query window with the following query:

```

/***** Script for SelectTopNRows command from SSMS *****/
SELECT [AccountNumber]
,[AccountDescription]
,[AccountCurrency]
,[AccountCreationDate]
,[AccountBalance]
,[AccountUserId]
FROM [GX_KB_DemoEbankingGXtest].[dbo].[Account]

```

Below the query, the 'Results' tab shows the following data:

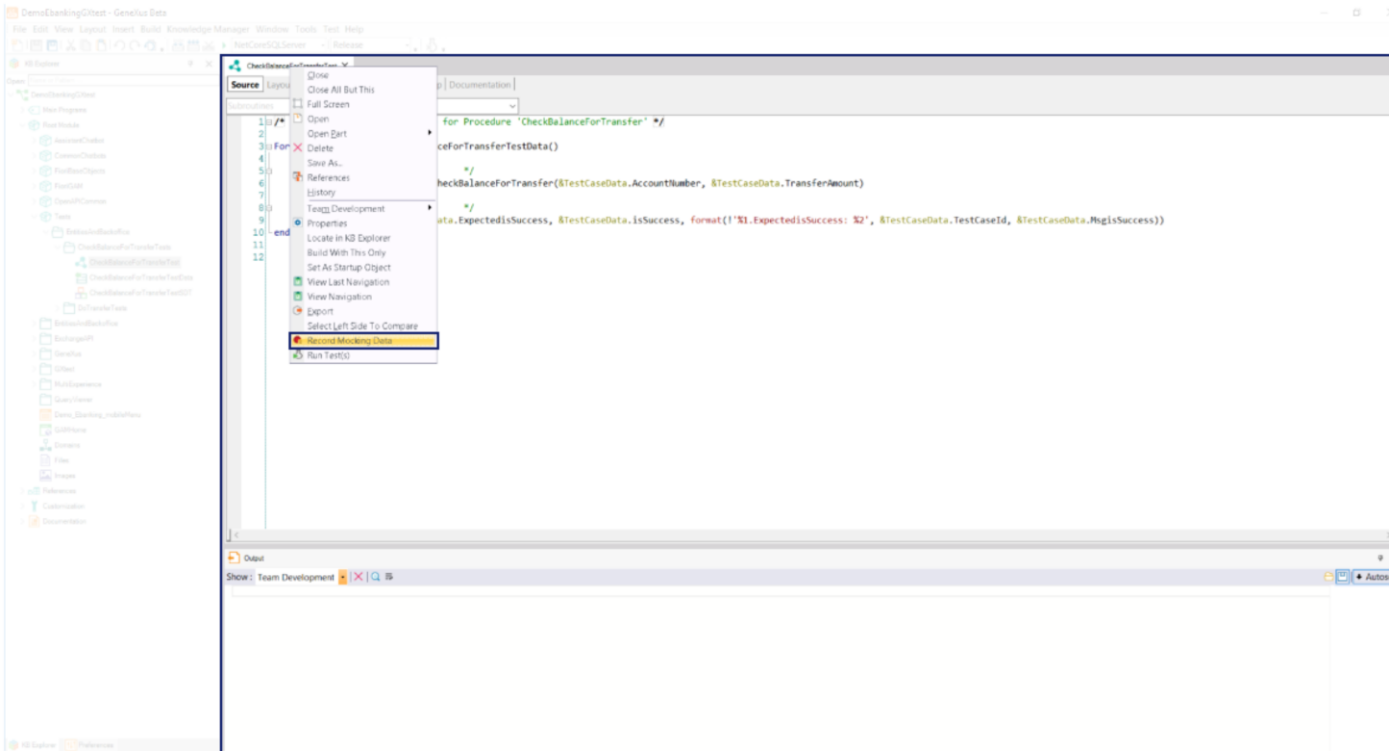
AccountNumber	AccountDescription	AccountCurrency	AccountCreationDate	AccountBalance	AccountUserId
1	5	Savings	2022-06-18 22:34:09.000	0.00	6
2	6	Savings	2022-06-18 22:34:09.000	100000.00	7

The status bar at the bottom indicates: 'Query executed successfully. EC2AMAZ-13038FJ\SQLEXPRESS ... Gtest (71) GX_KB_DemoEbankingGXtest 00:00:00 2'.

So, to use these test cases we need to set the database status.

In this case we need the AccountNumber 5 with balance 0, the AccountNumber 6 with balance enough to transfer 500 dollars and that AccountNumber 8 doesn't exist in database.

You can see the current database status of Account table.



Select the option “Record Mocking Data” doing right click over the unit test tab.

The screenshot displays the GeneXus IDE interface. On the left, a tree view shows the project structure, including folders like 'Main Programs', 'Root Module', and 'Tests'. The 'Tests' folder is expanded, showing a file named 'CheckBalanceForTransferTest'. The main window is titled 'Tests Results' and shows the execution details for 'CheckBalanceForTransferTest'. It indicates that the test started at 09:21:20, ended at 09:21:21, and elapsed 968 ms. The test results show 1 test passed successfully. Below the test results, the 'Output' window displays the following text:

```

Show: @Test
***** Run Tests started *****
GXtest components versions => Extension: 4.17.11.21376, Module: 4.17.11.21383
Execution data received C:\Models\demoBanking\gxtest\gxtestexecutionData.json...
STARTING unit test Tests.CheckBalanceForTransferTest...
RIMBO unit test Tests.CheckBalanceForTransferTest. Result: OK. Elapsed: 668 ms
*****
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\demoBanking\gxtest\NetCoreSQLServer\904\uebs\gxtest\tracefile_20220919_092119.gud'.
Success: Run Tests
3 sentences recorded
Mock File object 'Tests_CheckBalanceForTransferTest_MockDatafilelet' created

```

On the right side of the IDE, there is a 'Unit test execution details' window showing a table with columns 'Received', 'Obtained', and 'File'.

Received	Obtained	File
false	false	1 SentadoSuccess
true	true	2 SentadoSuccess
false	false	3 SentadoSuccess

This operation runs the test and saves the database sentences in a KB file. You can see the database sentences quantities and the Mock File name in the GXtest tab of the GeneXus Output.

From now on, when we run the unit test it will run using the recorded data instead of the database. That means that if the database changes, it doesn't affect the unit test results because the mock file contains all the data we need to execute the unit test.

DemoBankingGTest - GeneXus Data

File Edit View Layout Build Knowledge Manager Window Tools Test Help

TestCore03Server - Release

KB Explorer 9 X

- Open: DemoBankingGTest
 - Main Programs
 - Root Module
 - AssistantChatBot
 - CommonChabots
 - FinInBaseObjects
 - FinInSAM
 - OperAPI/Common
 - Tests
 - EntireAreaBackoffice
 - CheckBalanceForTransferTests
 - CheckBalanceForTransferTest
 - CheckBalanceForTransferTestData
 - CheckBalanceForTransferTestGDT
 - DoTransferTests
 - EntireAreaBackoffice
 - ExchangeAPI
 - GeneXus
 - GTest
 - MultiExperience
 - QueryViewer
 - Demo_Banking_mobileMenu
 - GAMHome
 - Genoma**
 - Files**
 - Imports
 - Referencia
 - Customization
 - Documentation

Test Results X CheckBalanceForTransferTest X Files X

Name Module Root Module

#	Name	Module	Description	Modified Date	Last User	Import Date	Last Build Date
1	Tests_CheckBalanceForTransferTest_MockDataFileList	Root Module	Tests_Check Balance For Transfer Test...	9/19/2022 9:21 AM	EC2AMAZ-1303BF/Administrator		
2	72-Rgd	Root Module	72-Rgd	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
3	OPTIMA_6_TTF	Root Module	OPTIMA_6_TTF	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
4	PTSana_Solid_rf	Root Module	PTSana_Solid_rf	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
5	PTSana_Relic_rf	Root Module	PTSana_Relic_rf	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
6	PTSana_Regular_rf	Root Module	PTSana_Regular_rf	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
7	72-Regular	Root Module	72-Regular	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		
8	SAP-Icons	Root Module	SAP-Icons	9/8/2022 8:56 PM	EC2AMAZ-1303BF/Administrator		

Search [] New

Output

```

Demo: GTest
----- Run Tests Started -----
GTest components version => Extension: 4.17.11-1176, Module: 4.17.11-1185
Execution area received C:\Users\demo Banking\GTest\GTest\ExecutionArea...
STARTING unit tests Tests_CheckBalanceForTransferTest...
INFO unit tests Tests_CheckBalanceForTransferTest_Results OK - elapsed: 0.03 s
-----
Execution ended successfully
Coverage data file for this execution was saved in "C:\Users\demo Banking\GTest\GTestCore03Server\GTest\test\tracefile_20220929_082129.psd".
Success: Run Tests
3 sentences responses
Trace file object Tests_CheckBalanceForTransferTest_MockDataFileList created

```

You can see the created file going to Files in the KB Explorer window.

The screenshot shows the GeneXus IDE interface. The main window displays a table of test results for the 'CheckBalanceForTransferTest' module. A context menu is open over the table, highlighting the option to edit a file. The table contains the following data:

Name	Module	Modified Date	Last Use	Import Date	Last Build Date
Tests_CheckBalanceForTransferTest	Root Module	9/19/2022 9:21 AM	ECJAMAZ-13038F\Administrator		
Tests_CheckBalanceForTransferTest_MockDataFileNet...	Test_...	9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
72 Build		9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
OPTIMA_B_TTF	F12	9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
PTServ_Sold_of		9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
PTServ_Regular_of		9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
72 Regular		9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		
SAP-icone		9/9/2022 8:56 PM	ECJAMAZ-13038F\Administrator		

The context menu options include: Edit File Tests_CheckBalanceForTransferTest_MockDataFileNet..., Update File Tests_CheckBalanceForTransferTest_MockDataFileNet..., Save File Tests_CheckBalanceForTransferTest_MockDataFileNet content as..., Open, Open Part, Copy (Ctrl+C), Cut (Ctrl+X), Paste (Ctrl+V), Delete (Del), Save As..., References (Ctrl+F12), History (Ctrl+Shift+H), Team Development, Properties (F4), and Export (Select Left Side To Compare).

The Output window at the bottom shows the following test results:

```

Run Tests started
-----
Build components version => Release: 8.17.31.21878, Build: 4.17.31.21395
Execution data registered C:\Users\jgomez\OneDrive\My Documents\GeneXus\...
STARTING unit test Tests_CheckBalanceForTransferTest...
EMBED unit test Tests_CheckBalanceForTransferTest. Results: Ok. Classes: 0/0
-----
Coverage report success!
Coverage data file for this execution was saved in: C:\Users\jgomez\OneDrive\My Documents\GeneXus\...
Success: Run Tests
Execution finished
Note: File object "Tests_CheckBalanceForTransferTest_MockDataFileNet" created
  
```

Right-clicking and selecting the option “Edit File Tests_CheckBalanceForTransferTest_MockDataNet...” you can open the file in your installed text editor to see the stored sentences and their results.

```
{
  "_array": [
    {
      "Data": {
        "innerArray": [
          [
            5,
            "Jack",
            "Savings",
            0,
            "2022-06-18T22:34:09.000",
            0.0000,
            6,
            1
          ]
        ]
      },
      "Key": "5,false,SELECT TMI.[AccountNumber], T2.[UserName] AS AccountUserName, TMI.[AccountDescription], TMI.[AccountCurrency], TMI.[AccountCreationDate], TMI.[AccountBalance], TMI.[AccountId] AS AccountUserId",
      "KeyPattern": ""
    },
    {
      "Data": {
        "innerArray": [
          [
            6,
            "Mary",
            "Savings",
            0,
            "2022-06-18T22:34:09.000",
            100000.0000,
            7,
            1
          ]
        ]
      },
      "Key": "6,false,SELECT TMI.[AccountNumber], T2.[UserName] AS AccountUserName, TMI.[AccountDescription], TMI.[AccountCurrency], TMI.[AccountCreationDate], TMI.[AccountBalance], TMI.[AccountId] AS AccountUserId",
      "KeyPattern": ""
    },
    {
      "Data": {
        "innerArray": [
          [
            8,
            "John",
            "Savings",
            0,
            "2022-06-18T22:34:09.000",
            0.0000,
            9,
            1
          ]
        ]
      },
      "Key": "8,false,SELECT TMI.[AccountNumber], T2.[UserName] AS AccountUserName, TMI.[AccountDescription], TMI.[AccountCurrency], TMI.[AccountCreationDate], TMI.[AccountBalance], TMI.[AccountId] AS AccountUserId",
      "KeyPattern": ""
    }
  ],
  null
},
"_head": 0,
"_size": 3,
"_tail": 3,
"_version": 4
}
```

If you open the file, you can find a json structure with the database sentences.

During the recording, the database sentences are stored in order (order is relevant) so it is expected that the test will run the sentences in the same order that was recorded. Consequently, if over time the table navigations change or database sentences are different, mocking data will need to be recorded again.

The screenshot displays the GeneXus IDE interface during a unit test execution. The left sidebar shows the project structure, with the test file 'Tests.CheckBalanceForTransferTest.gtest' selected. The main workspace is divided into three sections: a top status bar showing 'Start: 2022-09-19 09:46:44 End: 09:46:45 Elapsed: 1 mcs'; a central 'Tests ran: 1' section with a green checkmark and 'Execution results' showing 'Tests.CheckBalanceForTransferTest (873 ms)'; and a 'Unit test execution details' panel on the right. This panel contains a table with the following data:

Repeated	Obtained	Info
false	false	1 Expected:Success
true	true	2 Expected:Success
false	false	3 Expected:Success

At the bottom, the 'Output' window shows the execution log:

```

Set up succeeded
***** Run Tests started *****
Gitest components version => Gitest: 4.17.11.21874, Module: 4.17.11.21888
Execution data received C:\Models\DemoBankingGitest\GitestExecutionData.json...
INFO: Mock -> 3 sentences loaded from '...\Tests.CheckBalanceForTransferTest_mockData.gtest'
STARTING unit test Tests.CheckBalanceForTransferTest...
INFO: unit test Tests.CheckBalanceForTransferTest. Result: OK. Elapsed: 873 ms
*****
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemoBankingGitest\NetCore\SQLServer\@User\GitestTraceFile_20220919_094643.gcd'.
Success: Run Tests
  
```

When you execute the Unit Test after recording, now with Mocking, you can see in the Output a message that shows how many sentences were loaded from Mock file.

In case the tests need some data that was not recorded during the recording stage, that query will use real database data sources to get the answer. This will run the test you've selected using your current data sources/databases.

So, ensure you have the database in the required state so your recording can be checked into GeneXus server and add value to your team.

The screenshot shows the GeneXus IDE interface. The main window is titled "KB Explorer" and displays a tree view of the project structure. The "Files" panel shows a table of modules with the following data:

Name	Module	Description	Modified Date	Last User	Import Date	Last Build Date
Tests_CheckBalanceForTransferTe...	Root Module	Tests_Check_Balance	9/19/2022 9:21 AM	EC2AMAZ-1303BFJAdm...		
??_Build	Root Module	??_Build	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
OPTIMA_B_TTF	Root Module	OPTIMA_B_TTF	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
PTServs_Bola_tf	Root Module	PTServs_Bola_tf	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
PTServs_Italic_tf	Root Module	PTServs_Italic_tf	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
PTServs_Regular_tf	Root Module	PTServs_Regular_tf	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
??_Regular	Root Module	??_Regular	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		
SAP-icons	Root Module	SAP-icons	9/9/2022 8:56 PM	EC2AMAZ-1303BFJAdm...		

The "Properties" panel on the right shows various configuration options for the selected test. The "Mock Data" property is set to "True".

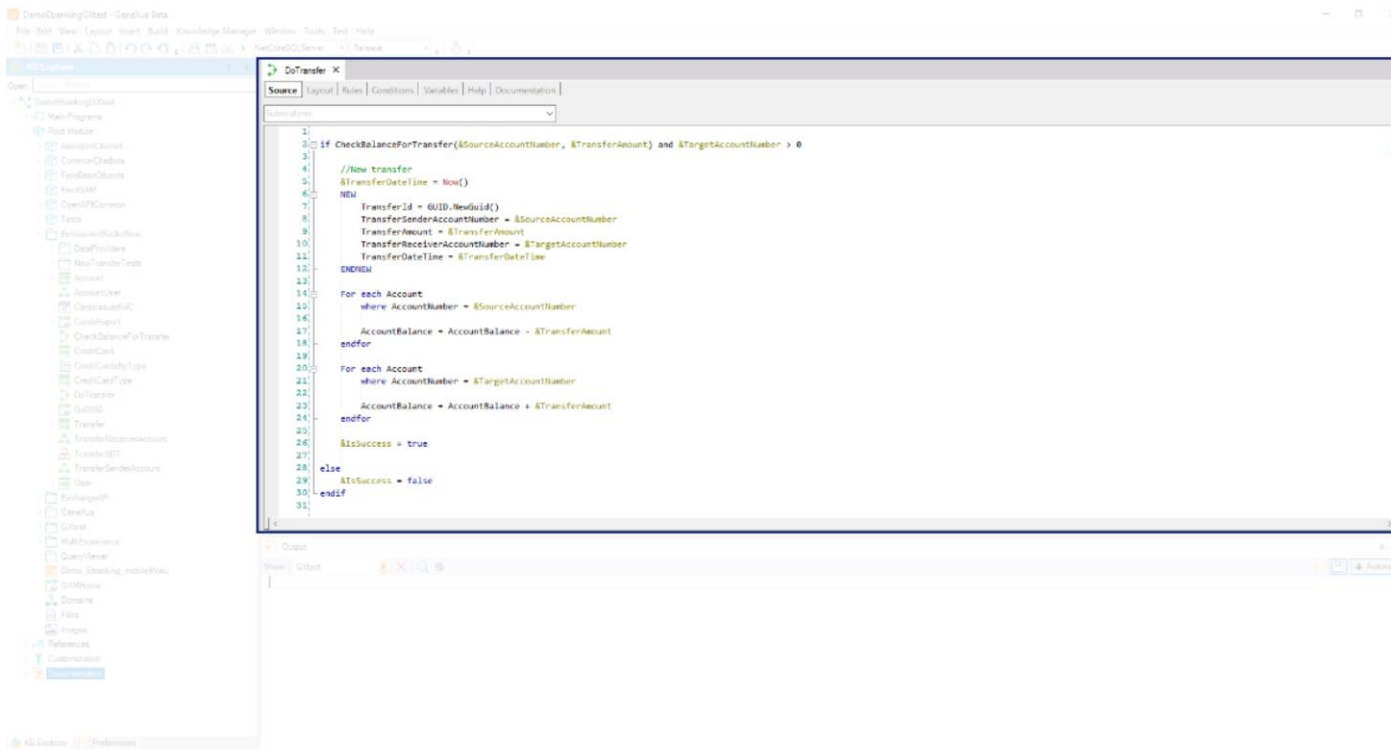
Note that the unit test properties were updated, the test has the “Mock Data” property in True.

Now, you can Commit the Unit Test and the Mock File to execute using the file in other environments and not have to set the database each time to execute the test over the procedure that you need to test.

Dynamic data management



Let's see how to manage dynamic data in the recorded Mock files.



Now that we know the Mocking DB concept, let's see an example of a recording with dynamic data.

We will mock the DoTransferTest unit test. It saves a new transfer in the table Transfer with TransferDateTime and TransferId being dynamic data.

First, we set the database status with the approved and rejected transactions, and we select the option "Record Mocking Data" to generate the Mocking file.

The screenshot displays the GeneXus IDE interface. On the left, a project tree shows the structure of the application, including modules like 'Main Programs', 'Root Module', and 'Tests'. The main editor window is titled 'DoTransferTestData' and contains the following code:

```
1) //rejected transfer
2) DoTransferTestSDT
3) {
4)   TestCaseId = '1'
5)   SourceAccountNumber = 5
6)   TransferAmount = 500
7)   TargetAccountNumber = 6
8)   ExpectedIsSuccess = false
9)   MsgIsSuccess = ''
10) }
11)
12) //approved transfer
13) DoTransferTestSDT
14) {
15)   TestCaseId = '2'
16)   SourceAccountNumber = 6
17)   TransferAmount = 500
18)   TargetAccountNumber = 5
19)   ExpectedIsSuccess = true
20)   MsgIsSuccess = ''
21) }
22)
23)
24)
25)
```

Below the code editor, there is an 'Output' window showing the results of the test cases.

The test cases include an approved transfer and a rejected transfer.

The screenshot displays the GeneXus IDE interface during a unit test execution. The main window shows the test results for 'Tests DoTransferTest', which passed successfully. The test execution details indicate 100% coverage and an elapsed time of 730 ms. The test results table shows that all expected and obtained values are true, with 1 expected success and 2 expected successes.

Expected	Obtained	Info
True	True	1.ExpectedSuccess
True	True	2.ExpectedSuccess

The console window at the bottom shows the following output:

```
Show: OKtest
Set up succeeded
----- Run Tests started -----
GXtest components versions -> Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemoBankingGXtest\GXtest\ExecutionData.json...
STARTING unit test Tests.DoTransferTest...
ENDED unit test Tests.DoTransferTest. Result: OK. Elapsed: 730 ms
-----
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemoBankingGXtest\NetCore\SQLServer\004\ueb\gctestTraceFile_20220919_123651.gpd'.
Success: Run Tests
5 sentences recorded
Mock File object 'Tests_DoTransferTest_MockDataFileMock' created
```

After we select “Record Mocking Data”. As you can see, it has recorded 5 database sentences.

Let’s run the Unit Test using the Mock File just recorded as it is.

The screenshot shows the GeneXus IDE interface during a test execution. The top pane displays 'Test Results X' for 'DoTransferTest.X'. The test 'Tests.DoTransferTest' is marked as successful with 100% coverage. The output pane shows the following log:

```

Set up succeeded
***** Run Tests started *****
GKtest components versions -> Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemobankingGKtest\GKtestExecutionData.json...
Info: Mock -> 5 sentences loaded from ".\Tests.DoTransferTest_mockData.gstest"
STARTING unit test Tests.DoTransferTest...
warning: Mock -> mismatch:
warning: Mock -> Expected '4,500,5,9/19/2022 3:16:52 PM,736815c-ff64-4b2a-acc0-d649181a4a3,True,INSERT 2070 [Transfer][TransferSenderAccountNumber], [TransferAmount], [TransferReceiverAccountNumber], [TransferDateTime], [Transac
warning: Mock -> Received '4,500,5,9/19/2022 3:19:27 PM,9b1c1687-d621-48cf-8ac8-d91c672586,True,INSERT 2070 [Transfer][TransferSenderAccountNumber], [TransferAmount], [TransferReceiverAccountNumber], [TransferDateTime], [Transac
warning: Mock -> results for THIS sentence will return from actual DB
warning: Mock -> mismatch:
warning: Mock -> Expected '4,500,5,9/19/2022 3:16:52 PM,736815c-ff64-4b2a-acc0-d649181a4a3,True,INSERT 2070 [Transfer][TransferSenderAccountNumber], [TransferAmount], [TransferReceiverAccountNumber], [TransferDateTime], [Transac
warning: Mock -> Received '500,5,True,UPDATE [Account] SET [AccountBalance]=[AccountBalance] + @0000TransferAmount WHERE [AccountNumber] = @000SourceAccountNumber'
warning: Mock -> results for THIS sentence will return from actual DB
warning: Mock -> mismatch:
warning: Mock -> Expected '4,500,5,9/19/2022 3:16:52 PM,736815c-ff64-4b2a-acc0-d649181a4a3,True,INSERT 2070 [Transfer][TransferSenderAccountNumber], [TransferAmount], [TransferReceiverAccountNumber], [TransferDateTime], [Transac
warning: Mock -> Received '500,5,True,UPDATE [Account] SET [AccountBalance]=[AccountBalance] + @0000TransferAmount WHERE [AccountNumber] = @000SourceAccountNumber'
warning: Mock -> results for THIS sentence will return from actual DB
ENDED unit test Tests.DoTransferTest. Result: OK. Elapsed: 749 ms
*****
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemobankingGKtest\NetCoreSQLServer\004\ubg\gktestTraceFile_20220919_123926.gud'.
Success: Run Tests

```

You can see warnings in the Output because a mismatch occurred between data recorded in the mock file and data requested during the test execution. As dynamic data is involved in the database operations, it changes in the execution and doesn't match with the data stored in the mock file. This happens when you have dynamic data for your sentences, let's say, an autonumber PK or a Date Time, for example.

In these cases, as you can read in the warning, results for the sentence will return from actual database. To take these results from the Mock File, it is necessary to modify the type of matching by setting the field `KeyPattern` while editing the mock File.

In this example, the mismatch is with the dynamic `DateTime` attribute of the table `Transfer`. So, let's modify the Mock File by adding regular expressions.

```

"_array": [
  {
    "Data": {
      "innerArray": [
        [
          5,
          "sack",
          "Savings",
          0,
          "2022-06-18T22:34:09.000",
          0.0000,
          0,
          1
        ]
      ]
    },
    "Key": "5,False,SELECT T1.[AccountNumber], T2.[UserName] AS AccountUserName, T1.[AccountDescription], T1.[AccountCurrency], T1.[AccountCreationDate], T1.[AccountBalance], T1.[AccountUserId] AS AccountUserId FROM ([Account] T1 INNER JOIN [User] T2 ON T1.[AccountUserId]=T2.[UserId]) WHERE T1.[AccountNumber]=5",
    "KeyPattern": ""
  },
  {
    "Data": {
      "innerArray": [
        [
          6,
          "Mary",
          "Savings",
          0,
          "2022-06-18T22:34:09.000",
          100000.0000,
          7,
          1
        ]
      ]
    },
    "Key": "6,False,SELECT T1.[AccountNumber], T2.[UserName] AS AccountUserName, T1.[AccountDescription], T1.[AccountCurrency], T1.[AccountCreationDate], T1.[AccountBalance], T1.[AccountUserId] AS AccountUserId FROM ([Account] T1 INNER JOIN [User] T2 ON T1.[AccountUserId]=T2.[UserId]) WHERE T1.[AccountNumber]=6",
    "KeyPattern": ""
  },
  {
    "Data": {
      "innerArray": [
        [
          1
        ]
      ]
    },
    "Key": "5,500,5,6/19/2022 3:36:52 PM,7364515c-ffe4-4b2e-aacc-6b0491854e3,True,INSERT INTO [Transfer]([TransferSenderAccountNumber], [TransferAmount], [TransferReceiverAccountNumber], [TransferDateTime], [TransferId], [TransferDetail]) VALUES(5,500,6,{\\$[1,2]/\\$[1,2]/\\$[2,4] \\$[1,2]/\\$[1,2]/\\$[1,2]}(PM/AM)),{\\$[0]-\\$[0]-\\$[0]-\\$[4]-\\$[12]}",
    "KeyPattern": "6,500,5,{\\$[1,2]/\\$[1,2]/\\$[2,4] \\$[1,2]/\\$[1,2]/\\$[1,2]}(PM/AM)),{\\$[0]-\\$[0]-\\$[0]-\\$[4]-\\$[12]}"
  },
  {
    "Data": {
      "innerArray": [
        [
          1
        ]
      ]
    },
    "Key": "500,6,True,UPDATE [Account] SET [AccountBalance]=[AccountBalance] - @AVS1TransferAmount WHERE [AccountNumber] = @AVS1SourceAccountNumber",
    "KeyPattern": ""
  },
  {
    "Data": {
      "innerArray": [
        [
          1
        ]
      ]
    },
    "Key": "500,6,True,UPDATE [Account] SET [AccountBalance]=[AccountBalance] + @AVS1TransferAmount WHERE [AccountNumber] = @AVS1SourceAccountNumber",
    "KeyPattern": ""
  }
]

```

IMPORTANT
"KeyPattern": "...{*regular expression*}..."

In the KeyPattern field of the Mock File it is possible to embed regular expressions so that they do not cause a mismatch when executing the test.

We suggest to copy and paste the value of the Key field in the KeyPattern field value, and then substitute the recorded DateTime and GUID for a regular expression that matches the value we expect.

The screenshot displays the Visual Studio IDE with the Test Results window open. The test 'Tests.DoTransferTest' has passed successfully. The output window shows the following details:

```
----- Run Tests started -----
GTest components versions -> Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemobankingGTest\GTest\executionData.json...
Info: Mock -> 5 sentences loaded from ".\tests.DoTransferTest_mockData.gtest"
STARTING unit test Tests.DoTransferTest...
ENDED unit test Tests.DoTransferTest. Result: OK. Elapsed: 564 ms
-----
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemobankingGTest\NetCoreSQLServer\004\web\gtestTraceFile_20220919_124908.gud'.
Success: Run Tests
```

The Test Results window shows the following table:

Expected	Obtained	Info
Info	Info	1.ExpectedSuccess
True	True	2.ExpectedSuccess

If we run the test again, the test executes the sentences read from the Mock File successfully.

The screenshot shows the GeneXus IDE interface. On the left, a tree view displays the project structure for 'DemoBankingQTest'. The main area shows a 'Commit' dialog with a comment field containing 'Unit Tests Mock updated and Database Mock Files'. Below the comment field is a table of 'Pending Commits (4/6)'. The table has columns for Name, Description, Modified On, Module, Local State, Last Synchronized, and User. The commits listed are:

Name	Description	Modified On	Module	Local State	Last Synchronized	User
DoTransferTest	Do Transfer Test	9/19/2022 12:36 PM	Tests	Modified	9/19/2022 11:53 AM	EC248642-13033F-Administrator
CheckBalanceForTransferTest	Check Balance For Trans	9/19/2022 9:21 AM	Tests	Modified	9/19/2022 9:20 AM	EC248642-13033F-Administrator
Tests_CheckBalanceForTransferTest_MockDataFile.txt	Tests_Check Balance F...	9/19/2022 9:21 AM	Test Module	Inserted	9/19/2022 2:04 PM	EC248642-13033F-Administrator
Tests_DoTransferTest_MockDataFile.txt	Tests_Do Transfer Test...	9/19/2022 12:47 PM	Test Module	Inserted	9/19/2022 2:04 PM	EC248642-13033F-Administrator

Below the table, there is a 'Choose' dialog box with the text 'SHOW: Team Development'.

Again, once you have your mock data ready, you can commit the tests with Mock property in True and the Mock files to GeneXus server and tests won't depend on the state of the database used by other developers. And, more importantly, its success won't depend on the different environments used by your continuous integration pipeline either.

If your test is successful in your local environment it will work the same in any other environment (as long as the version of the objects is the same).

Note that you will prefer database mocking in tests without database assertions, as in these examples, in which we checked the output parameters. When the test checks the impact in the real database you don't want to use the DataBase Mocking. In those scenarios it is desirable to have DB initialization scripts instead.

GeneXus[™]
by **Globant**

training.genexus.com
wiki.genexus.com