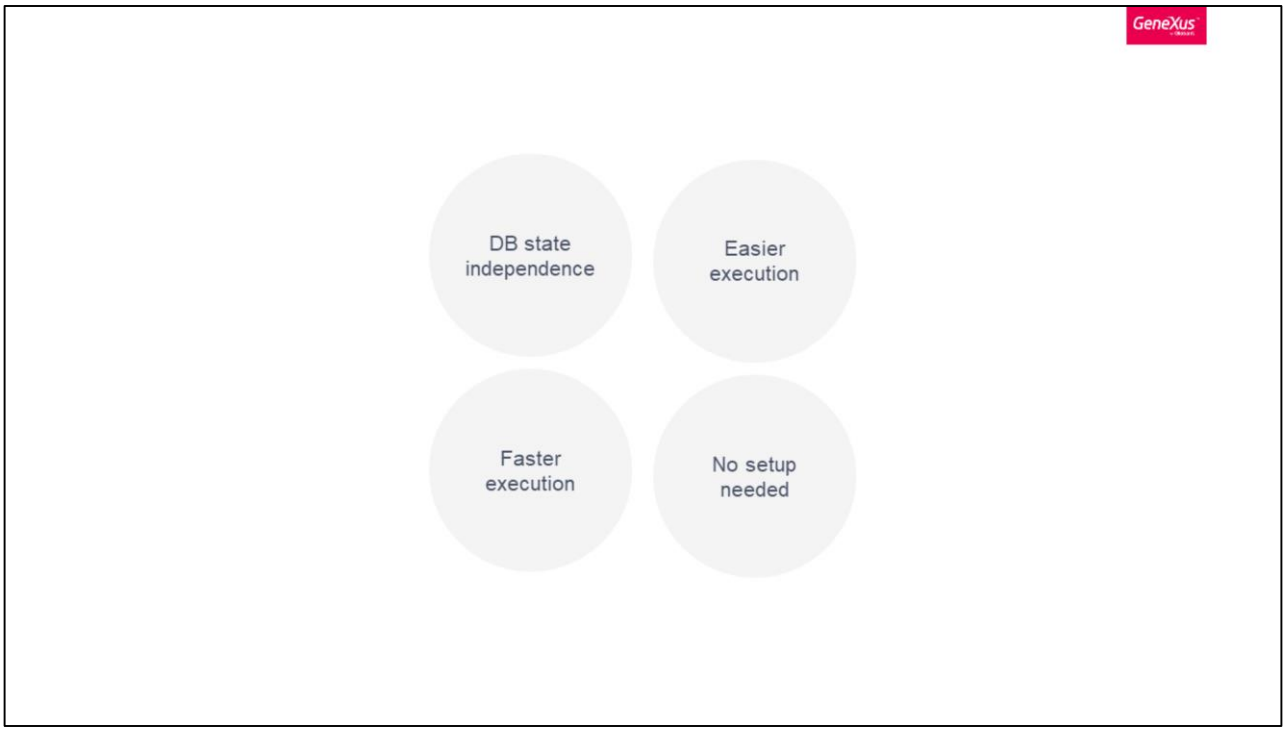

GeneXus™
by Globant

Database Mocking

GeneXus

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.

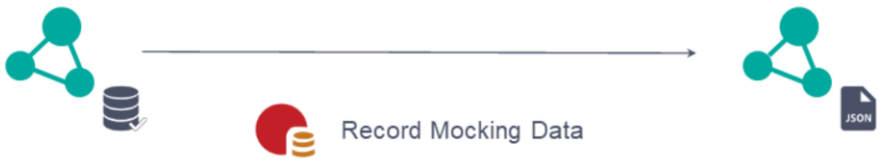
Database Mocking in GeneXus

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.



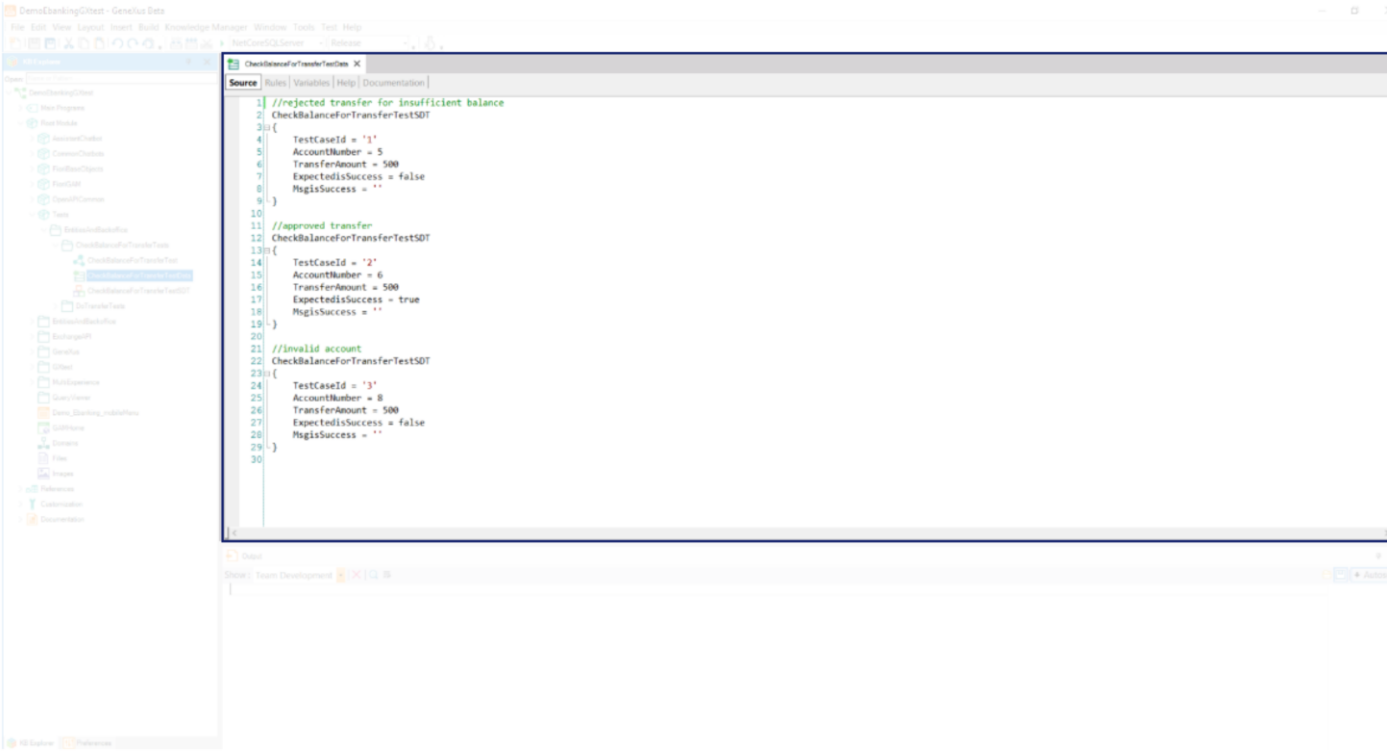
Mocking works by recording data (SQLs / results) used for a test by listening for all sentences and responses during the **Record Mocking Data operation**. In GeneXus, database sentences are stored in a json file (as **mocking data**). You will be able to save the set of data used in each unit test and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

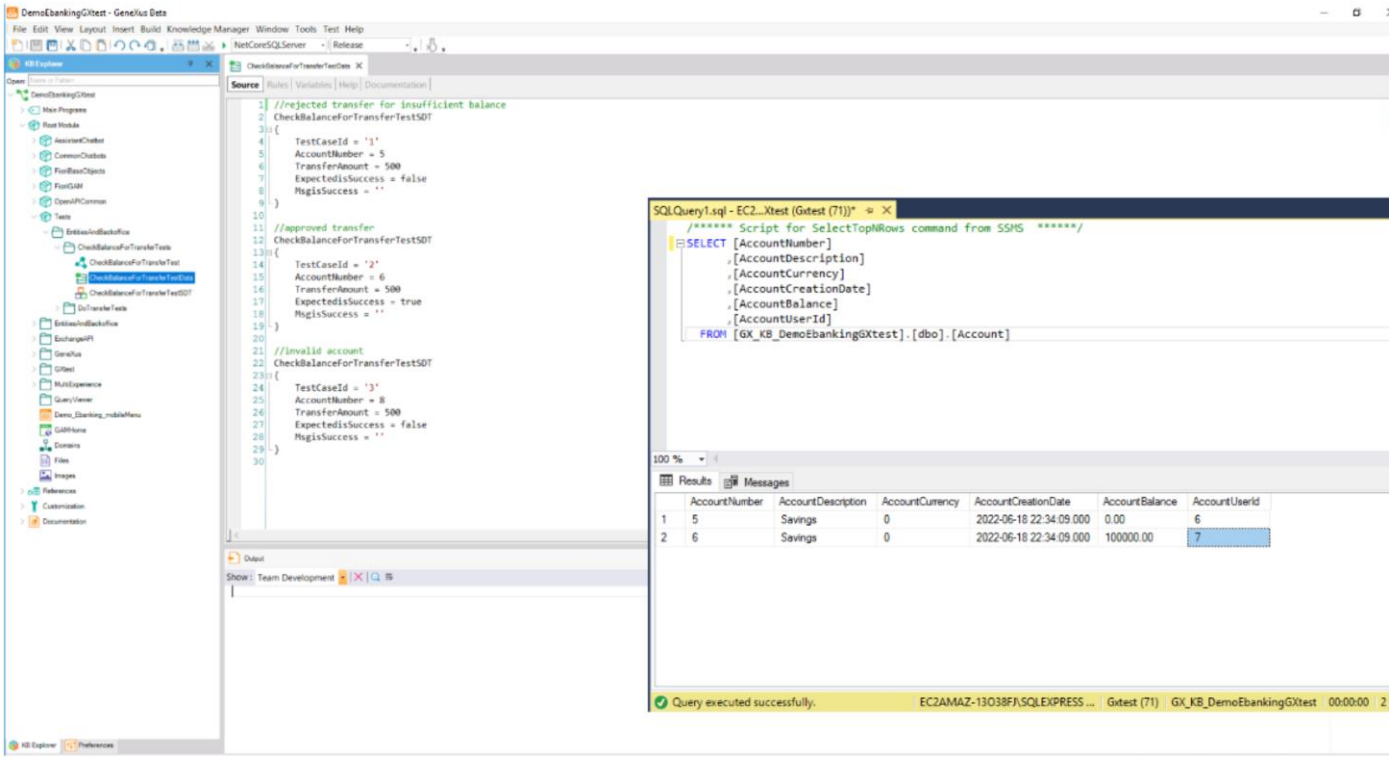
First Database Mocking example

GeneXus by Globant



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

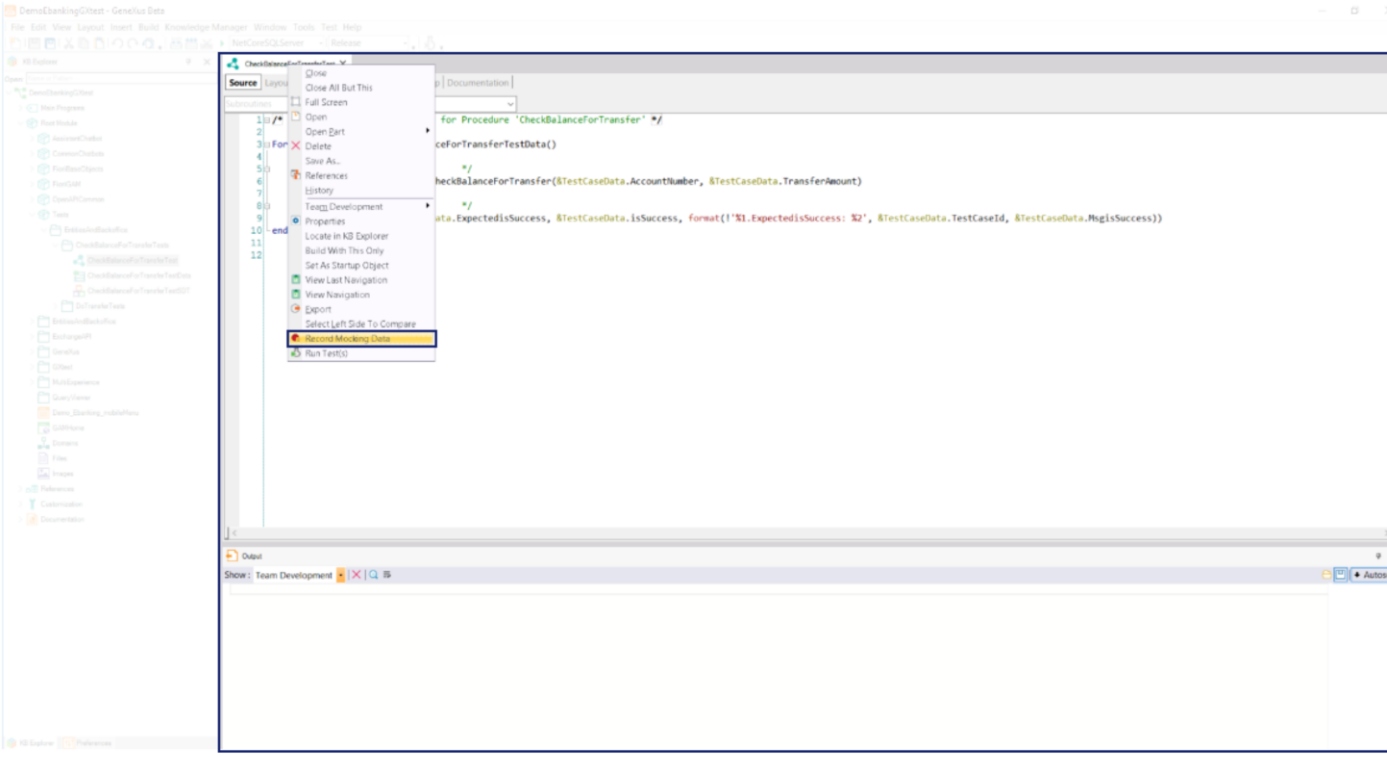
Remember that we tested this proc with three test cases: rejected transfer for not enough balance, an approved transfer and an invalid account.



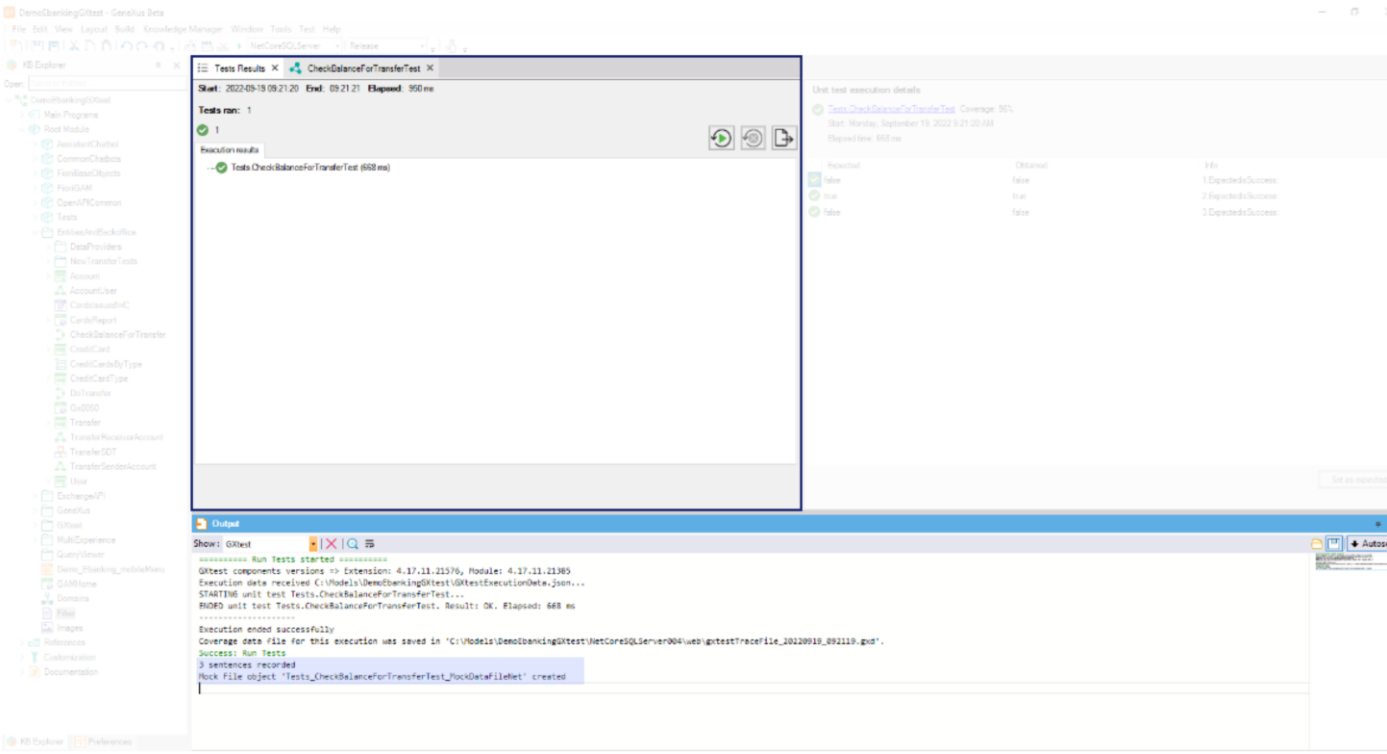
So, to use these test cases we need to set the database status.

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.

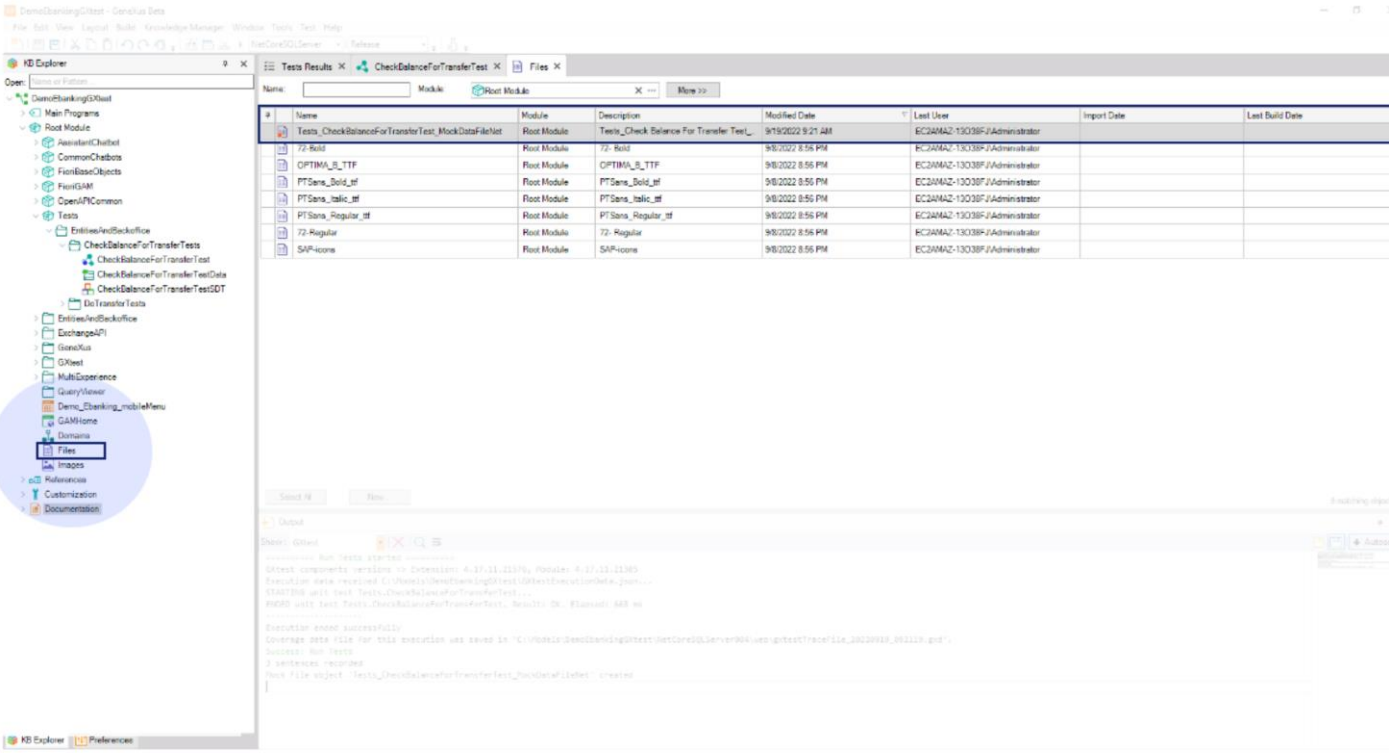


Select the option “Record Mocking Data” doing right click over the unit test tab.

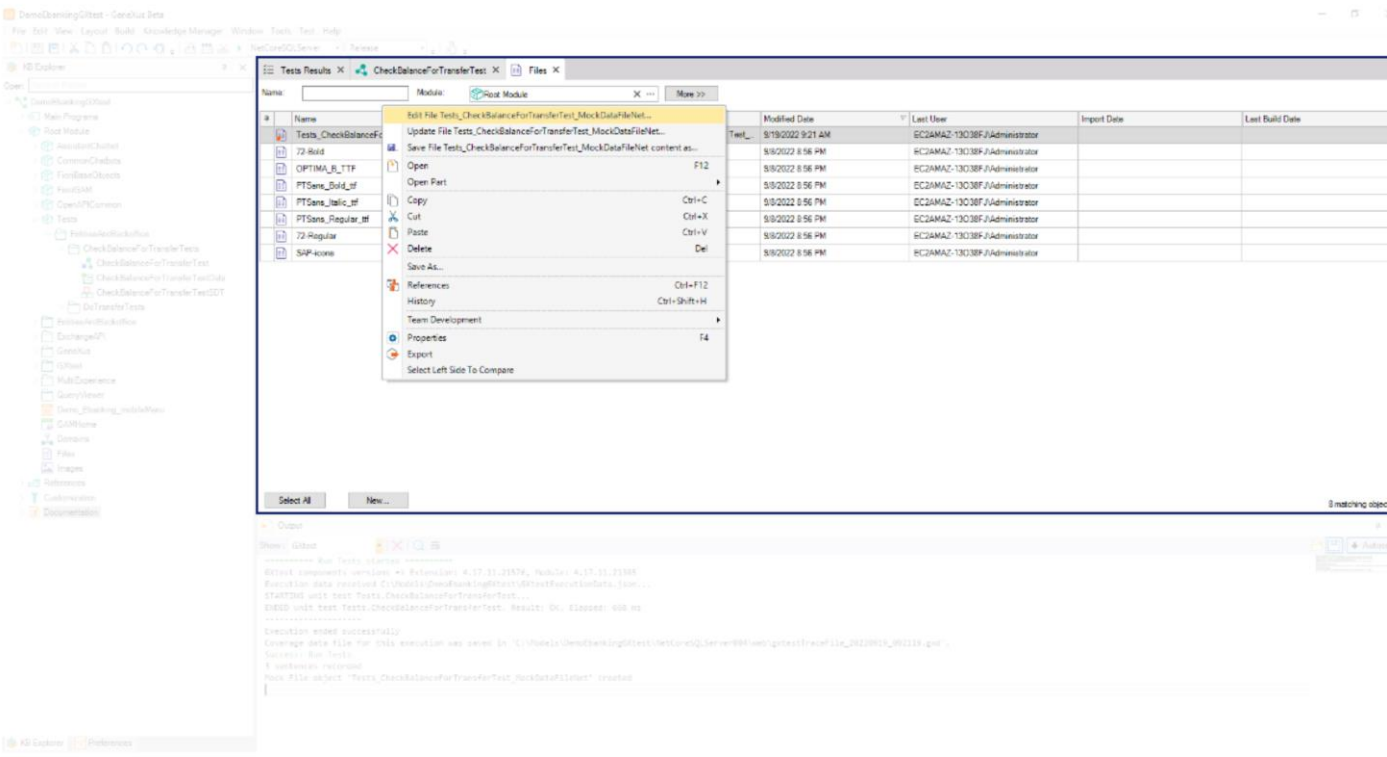


This operation runs the test and saves the database sentences in a KB file. You can see the database sentences quantities and the Mock File name in the GXtest tab of the GeneXus Output.

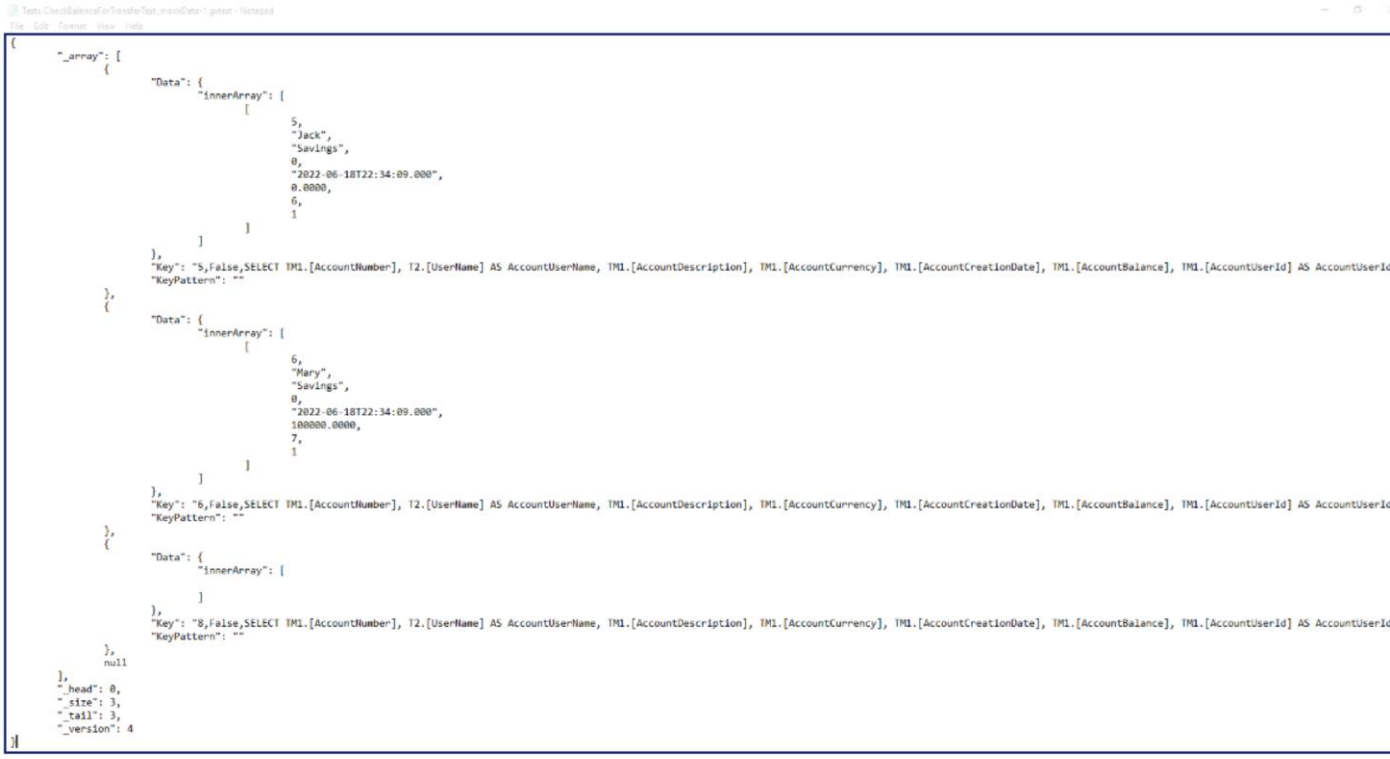
From now on, when we run the unit test it will run using the recorded data instead of the database. That means that if the database changes, it doesn't affect the unit test results because the mock file contains all the data we need to execute the unit test.



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

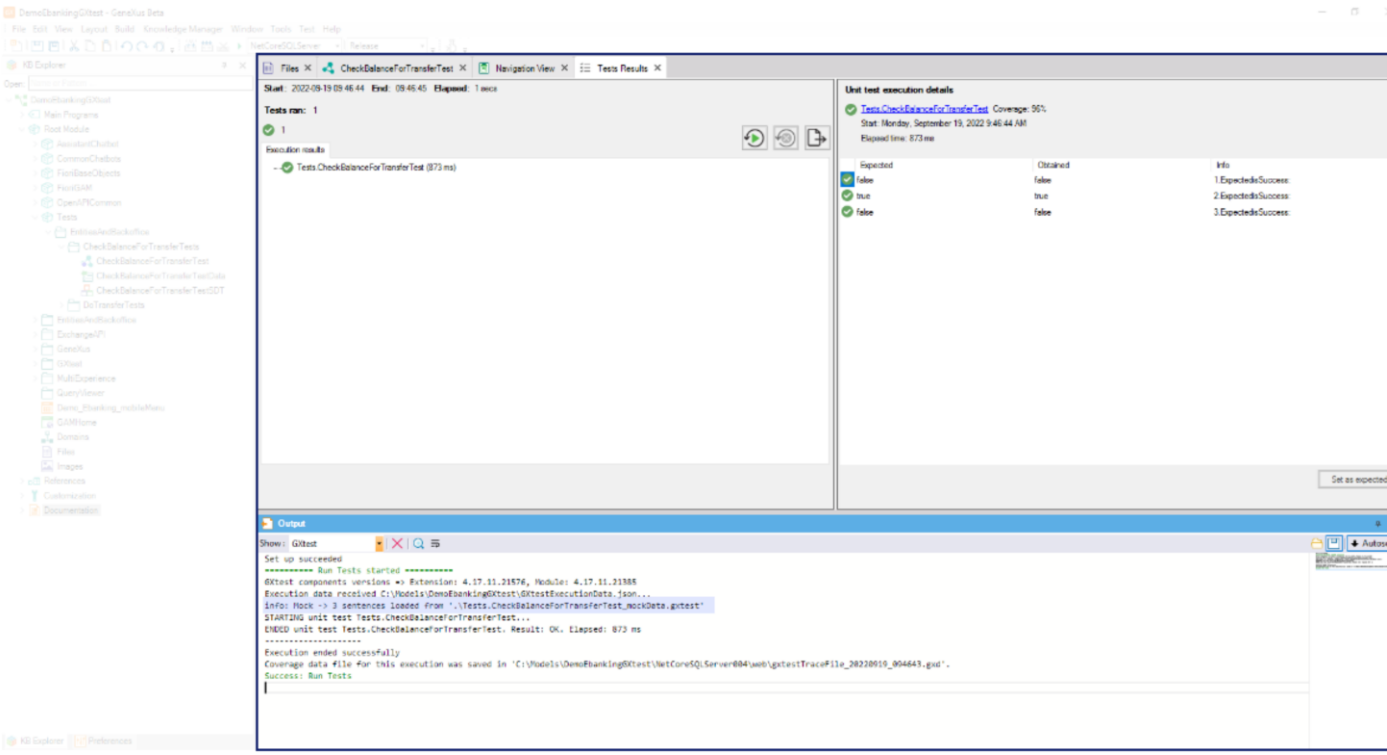


Right-clicking and selecting the option “Edit File Tests_CheckBalanceForTransferTest_MockDataNet…” you can open the file in your installed text editor to see the stored sentences and their results.



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

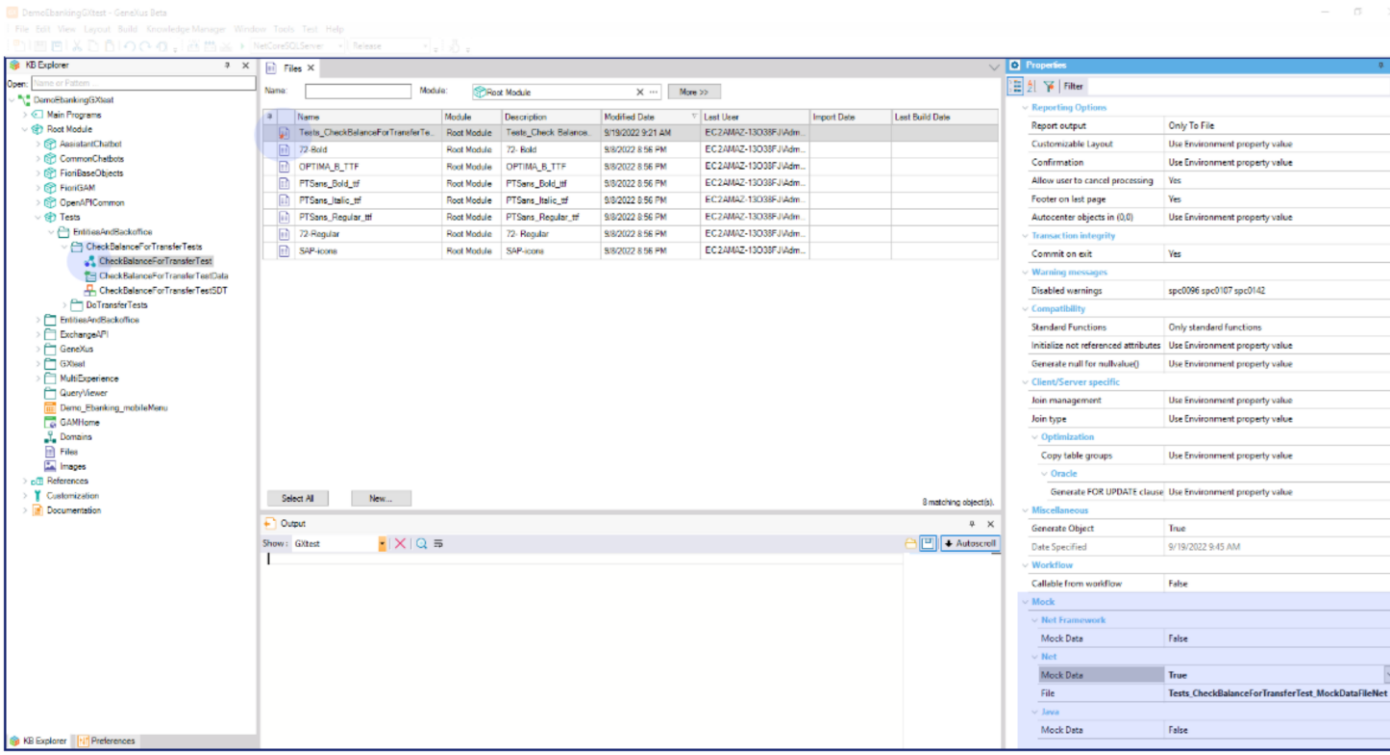
During the recording, the database sentences are stored in order (order is relevant) so it is expected that the test will run the sentences in the same order that was recorded. Consequently, if over time the table navigations change or database sentences are different, mocking data will need to be recorded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



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.

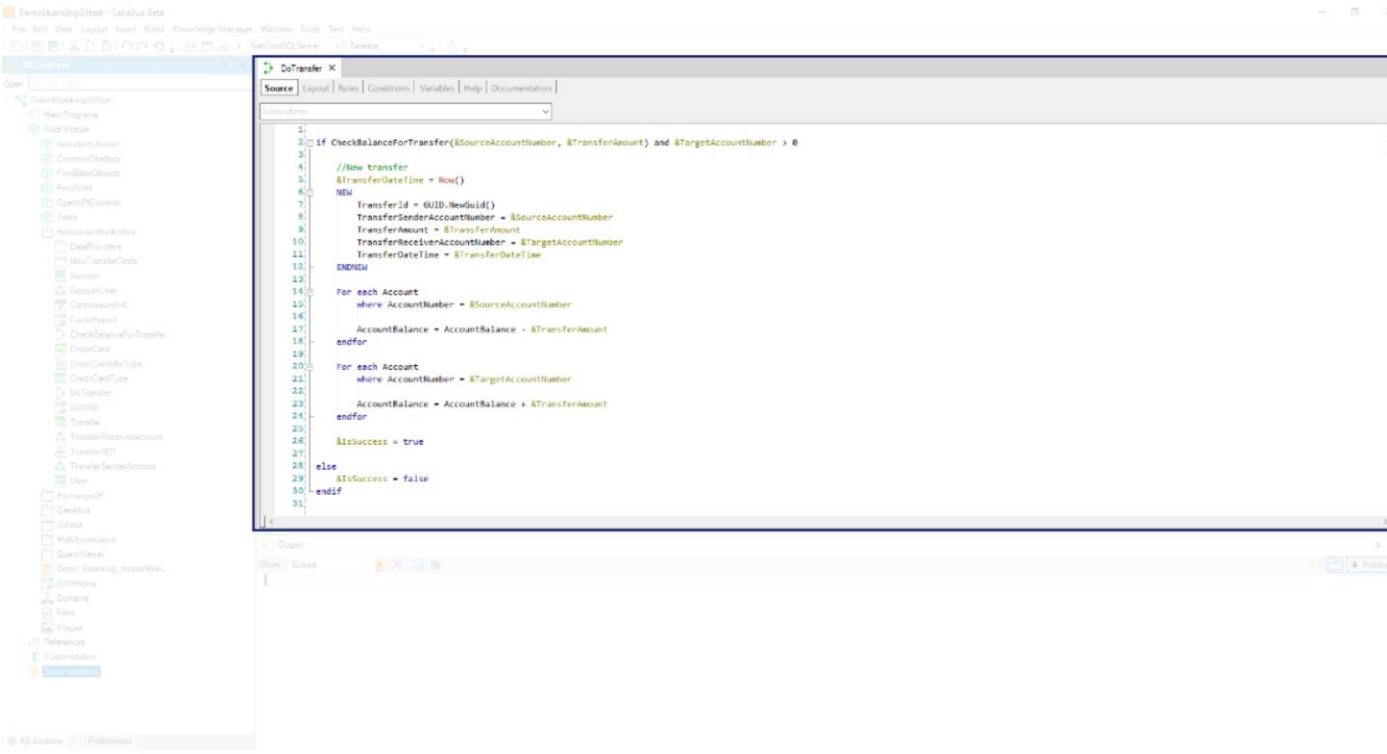
So, ensure you have the database in the required state so your recording can be checked into GeneXus server and add value to your team.



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.

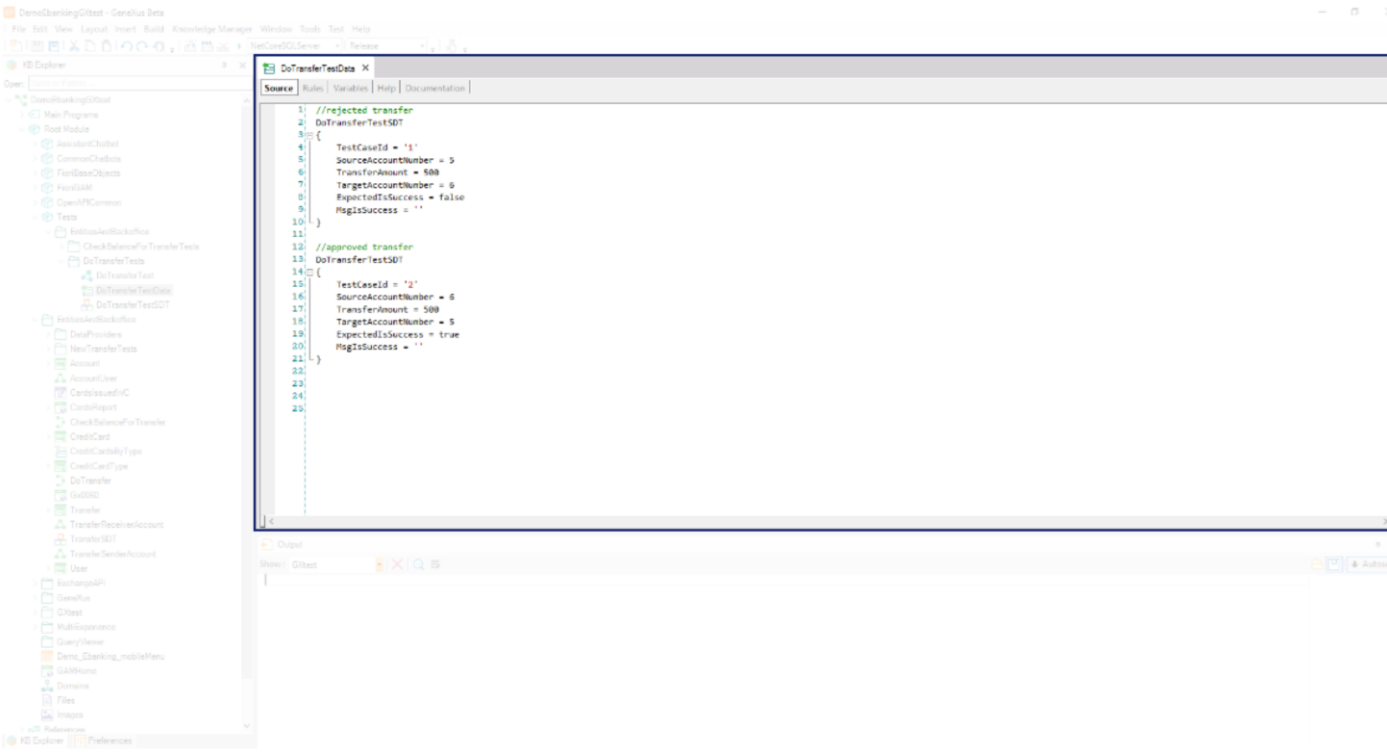
Let’s see how to manage dynamic data in the recorded Mock files.



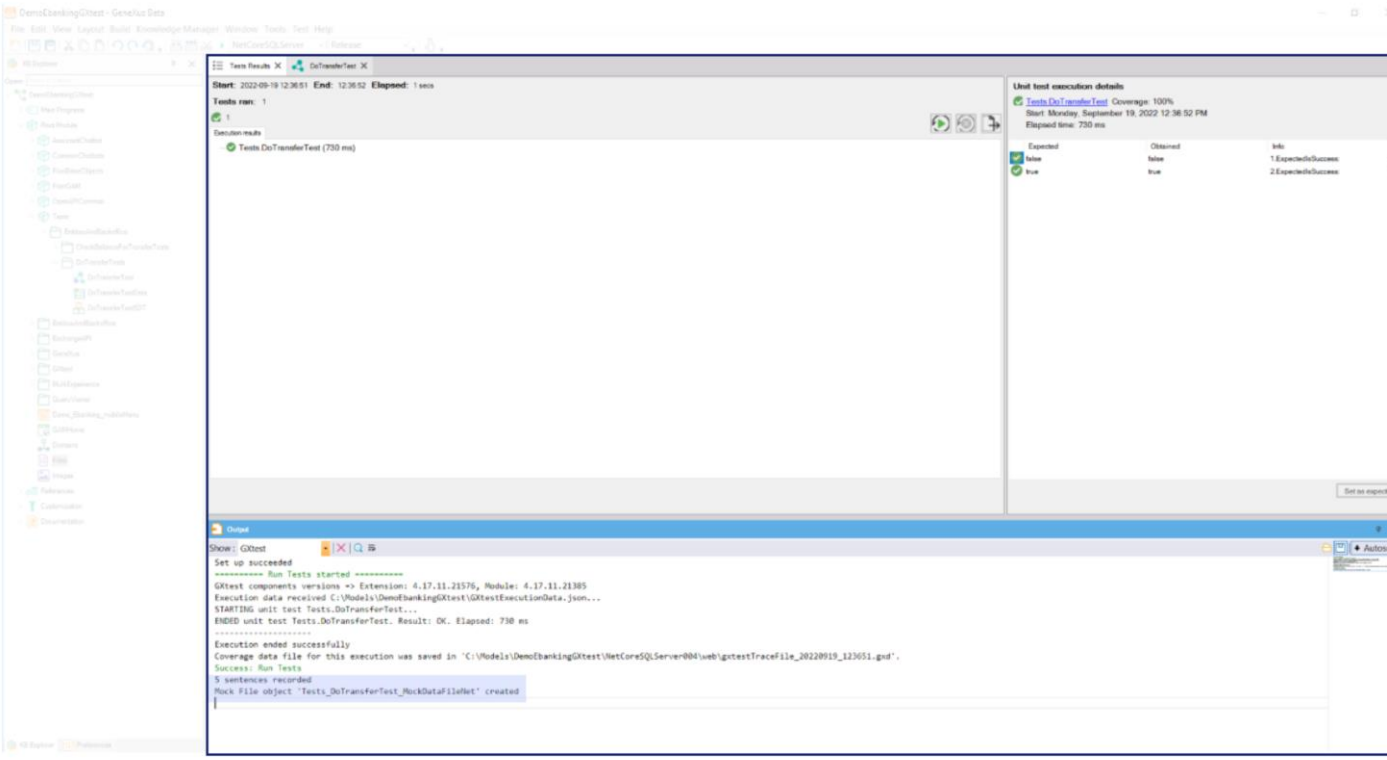
Now that we know the Mocking DB concept, let's see an example of a recording with dynamic data.

We will mock the DoTransferTest unit test. It saves a new transfer in the table Transfer with TrasferDateTime 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.

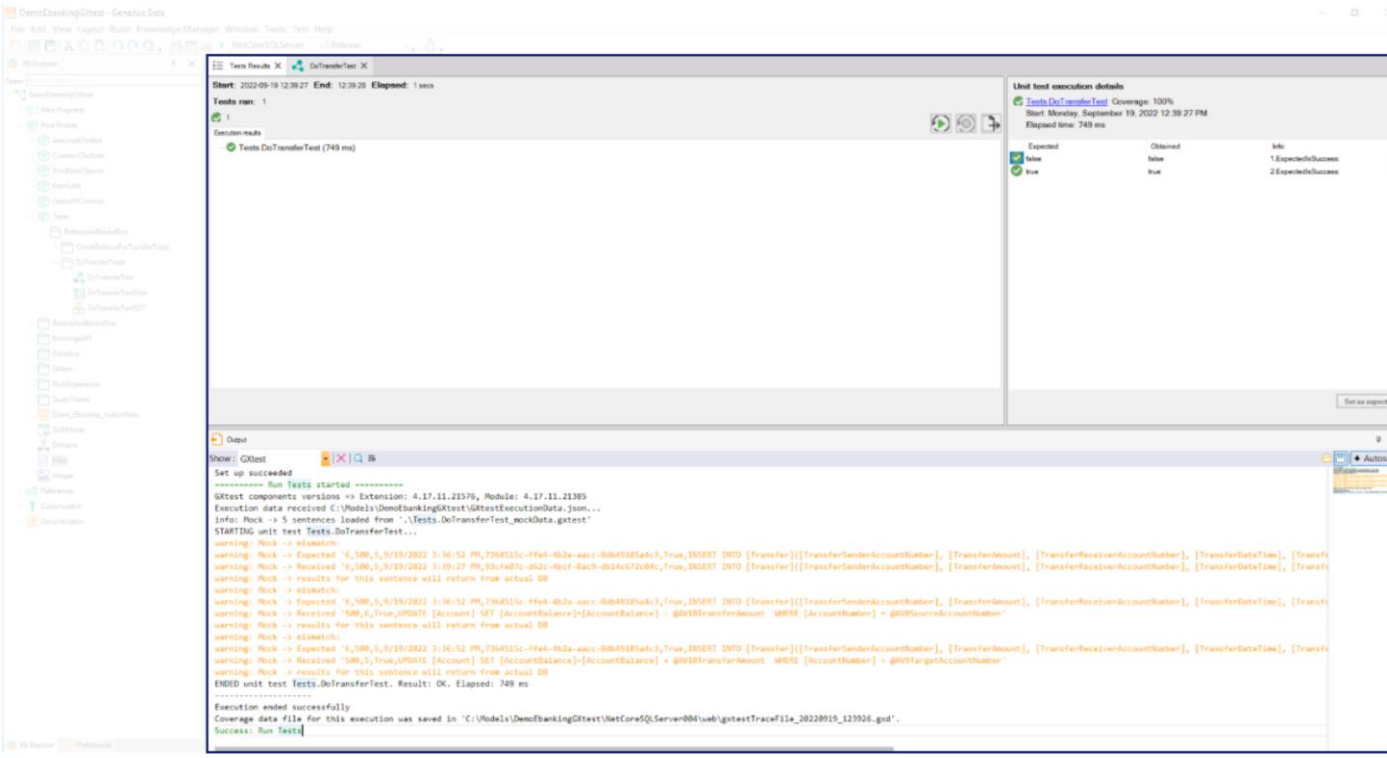


The test cases include an approved transfer and a rejected transfer.



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.



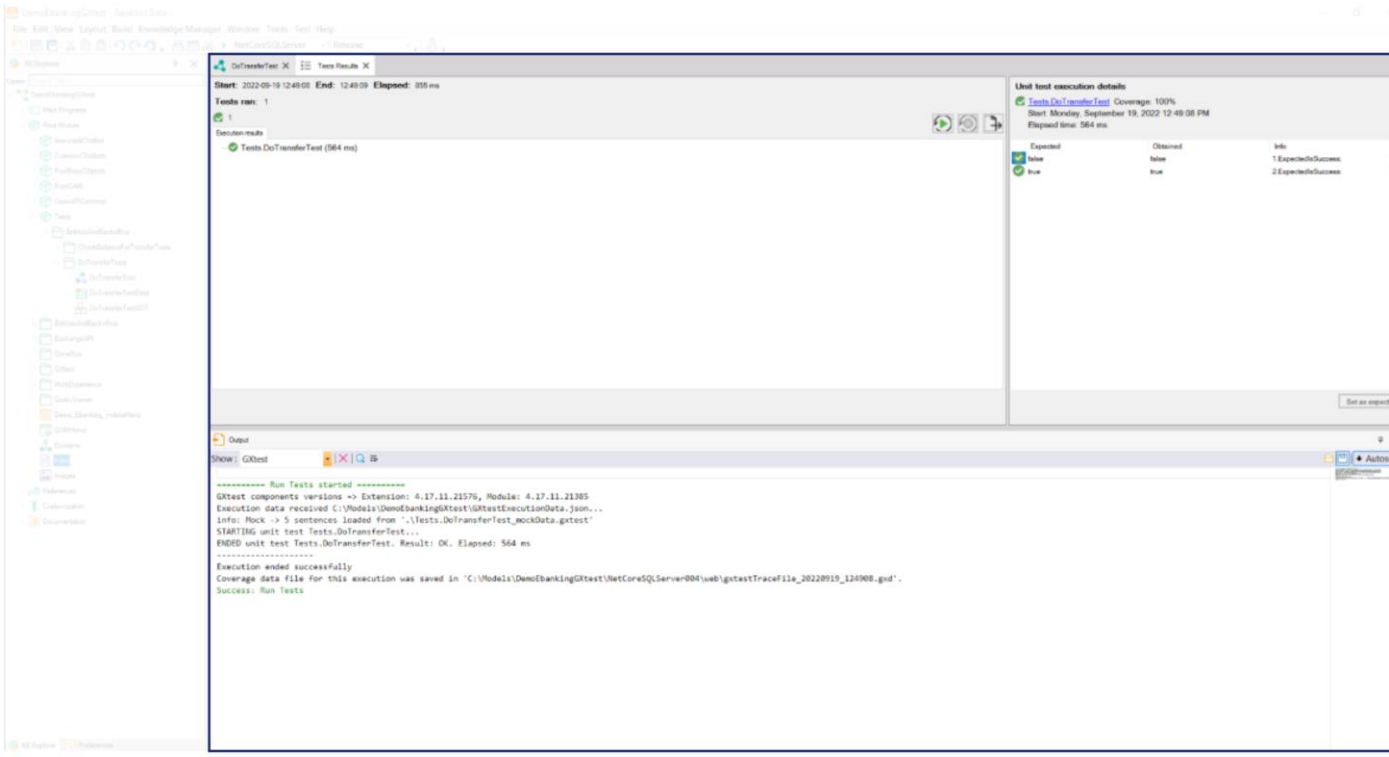
You can see warnings in the Output because a mismatch occurred between data recorded in the mock file and data requested during the test execution. As dynamic data is involved in the database operations, it changes in the execution and doesn't match with the data stored in the mock file. This happens 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.

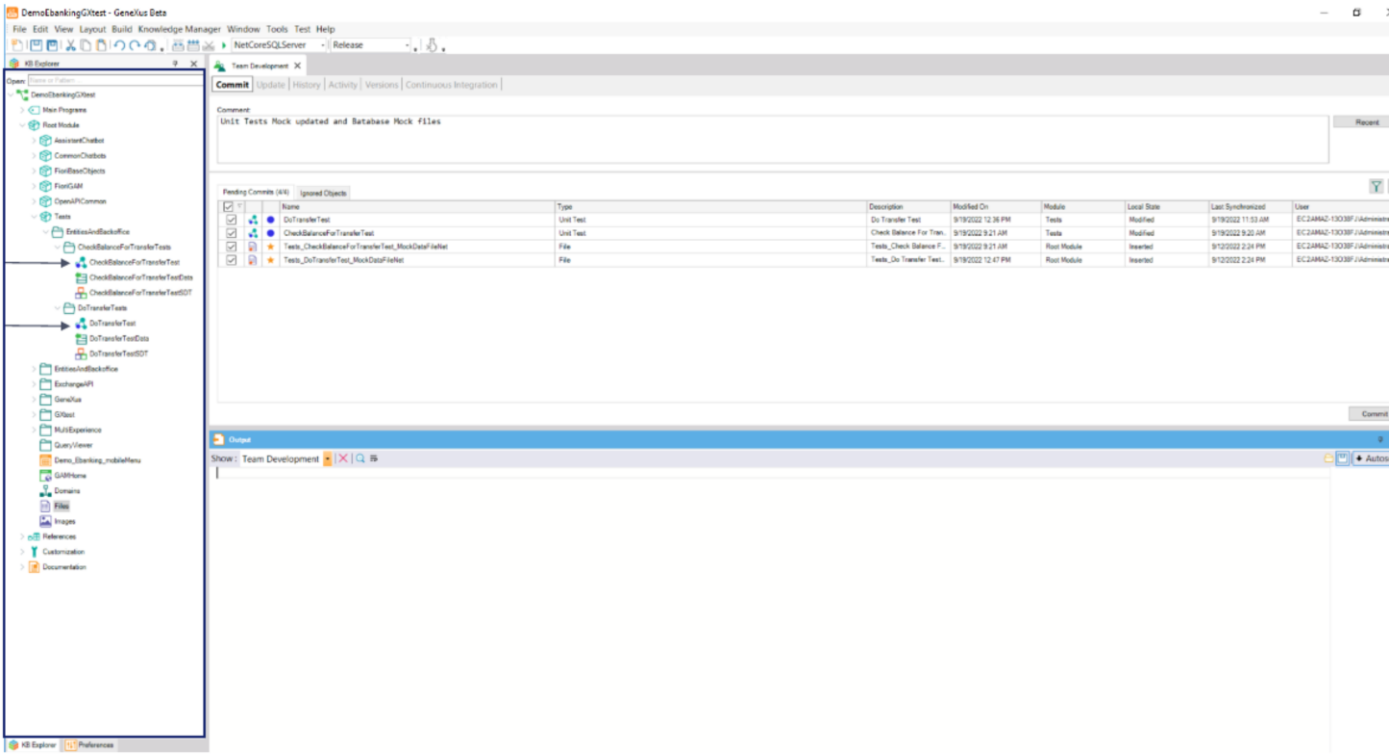
In this example, the mismatch is with the dynamic DateTime attribute of the table Transfer. So, let's modify the Mock File by adding regular expressions.

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.



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



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