

# Automatisation de tests via SoapUI

**Chantal Zimmer**

Mars 2023

SoapUI est un logiciel open source permettant les tester les API, mal connu et pourtant très utile pour effectuer des tests techniques et même automatiser des scénarios de bout en bout. Je vous propose une présentation du périmètre d'action du logiciel suivie de l'explication de l'architecture d'un projet d'automatisation pour finir par deux exemples concrets d'utilisation.

### Périmètre d'action du logiciel SoapUI

Le logiciel SoapUI permet de faire bien plus que de simples appels de web services SOAP ou REST.

Son périmètre d'actions comprend notamment:

- Appel de web services SOAP,
- Appel de web services REST,
- Requête en base de données,
- Transfert de variables,
- Script en langage Groovy,
- Assertion permettant de vérifier les données.

Cette liste est non exhaustive, personnellement j'ai encore beaucoup de fonctionnalités à expérimenter comme:

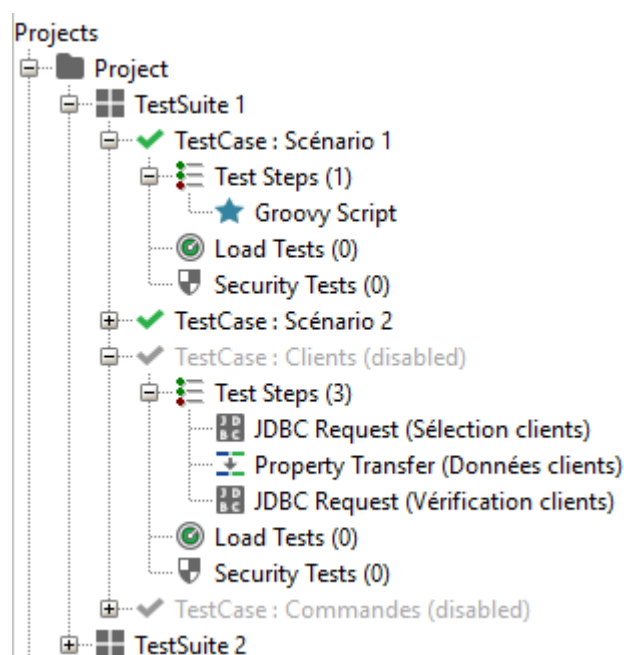
- Requête HTTP,
- Requête AMF,
- Conditional Goto,
- Protocole de messagerie MQTT

# Architecture du projet

Le projet d'automatisation pour être facilement maintenable se doit d'avoir une architecture claire et optimisée ainsi que des données variabilisées.

## a) Architecture

Pour nos projets, le choix s'est porté sur une architecture en briques. Cette organisation permet d'éviter au maximum la duplication des services, requêtes et scripts utilisés.



Le projet est composé d'un ou plusieurs TestSuite qui représentent le périmètre à tester.

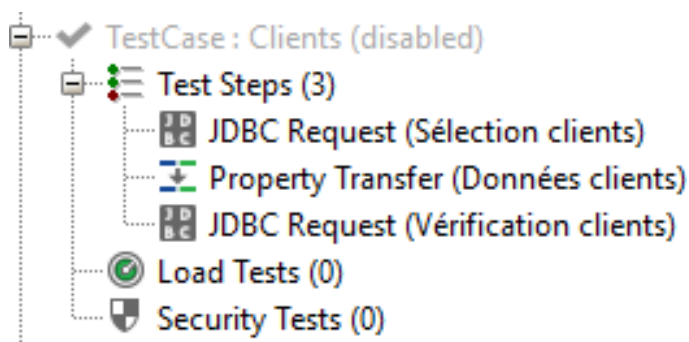
A l'intérieur du TestSuite, chaque TestCase représente une brique. Nous avons des briques actives et inactives:

- ✓ Les briques actives correspondent aux scénarios à exécuter.
- ✓ Les briques inactives regroupent par type de fonctionnalité les différents appels, requêtes et scripts utilisés par les scénarios.

Chaque brique active comprend un seul TestStep qui est un script Groovy permettant d'exécuter les différentes étapes du scénario.

```
Groovy Script
1 def project = testRunner.testCase.testSuite.project
2
3 ///////////////////////////////////////////////////Sélection de clients...//////////////////////////////////////
4 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Sélection clients)'] )
5 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['Property Transfer (Données clients)'] )
6 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Vérification clients)'] )
```

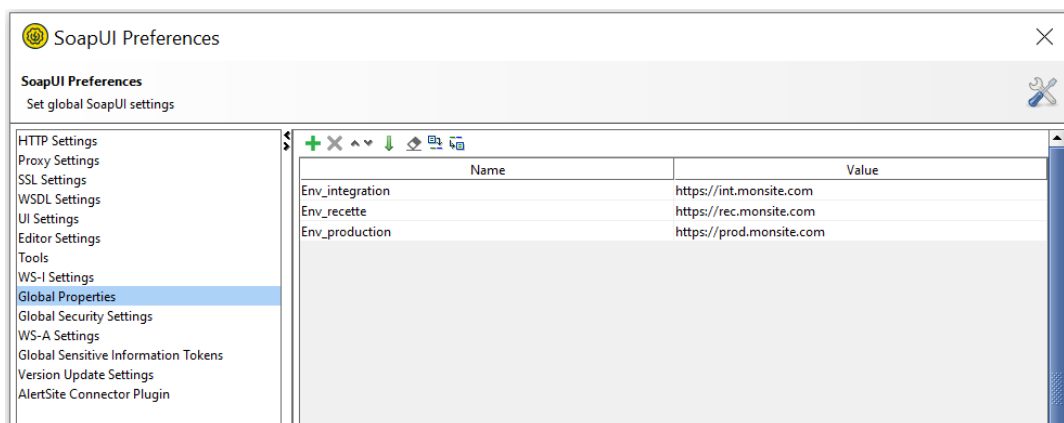
Chaque brique inactive comprend un ou plusieurs pas de test.



### b) Variabilisation

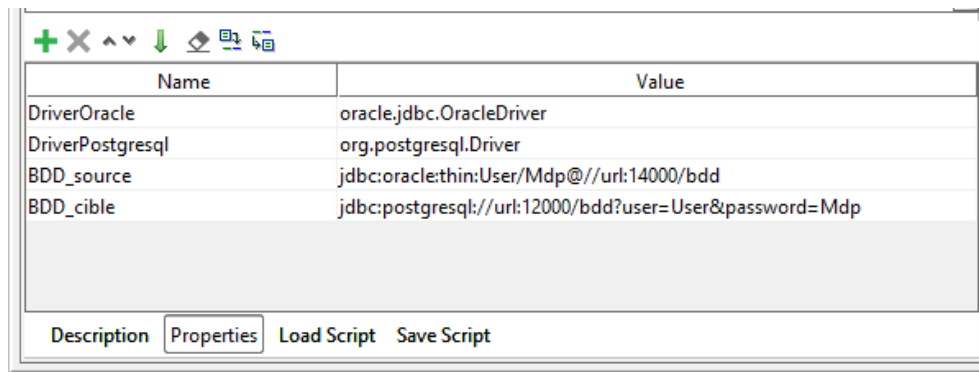
Au niveau de la variabilisation des données, nous avons opté pour l'enregistrement des variables à différents niveaux.

Les données communes à différents projets comme les url des différents environnements sur lesquels les tests doivent être exécutées sont stockées au niveau « Global Properties »



Les données communes à plusieurs périmètres comme les url de connexion aux bases de données ou les url des différents services se trouvent au niveau des « Properties du Project »

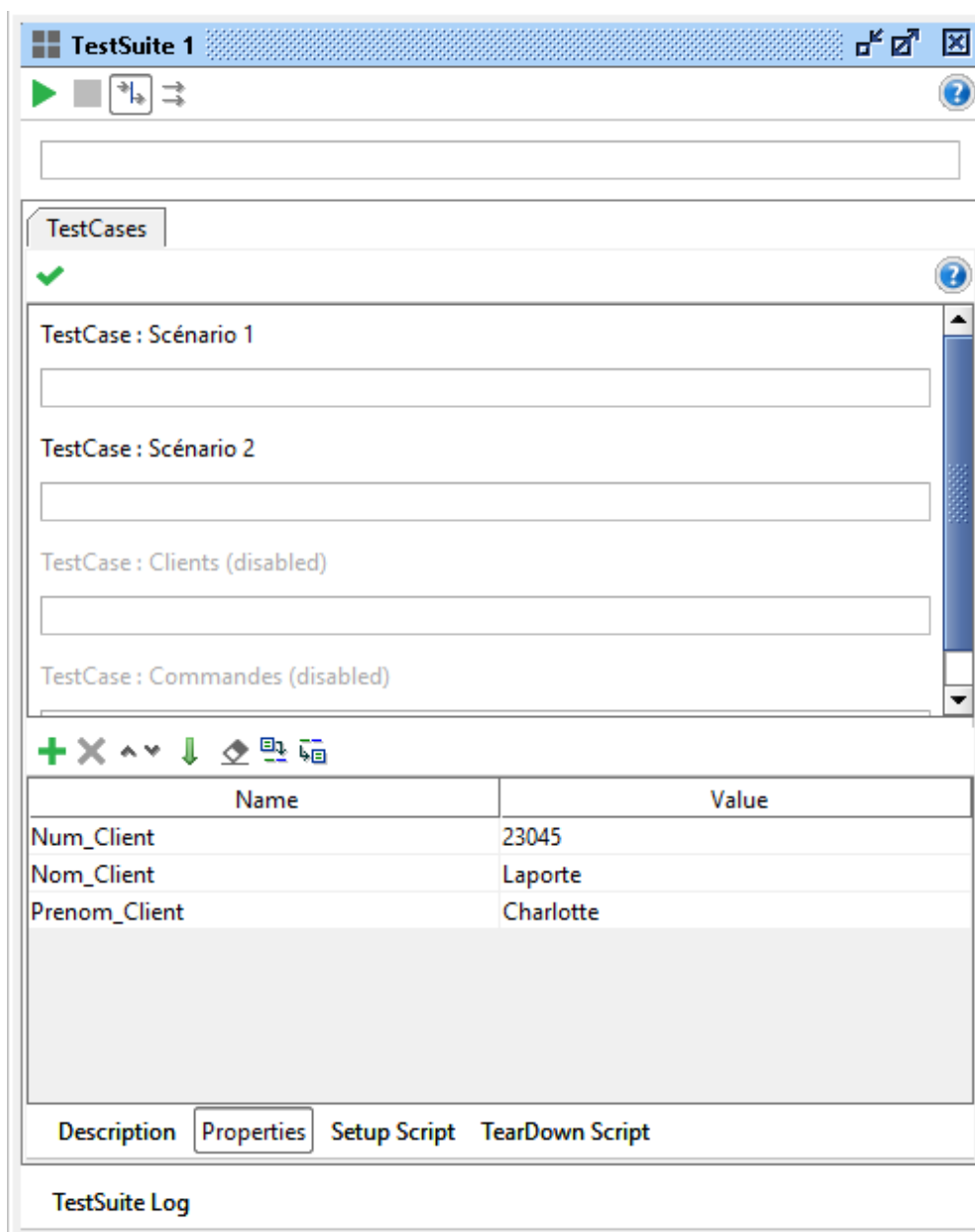
## Automatisation de tests via SoapUI



A screenshot of the SoapUI Properties dialog for a TestSuite. The dialog has a title bar with standard window controls. Below the title bar is a table with two columns: 'Name' and 'Value'. The table contains four rows of data. At the bottom of the dialog, there are four tabs: 'Description', 'Properties', 'Load Script', and 'Save Script'. The 'Properties' tab is currently selected.

Name	Value
DriverOracle	oracle.jdbc.OracleDriver
DriverPostgresql	org.postgresql.Driver
BDD_source	jdbc:oracle:thin:User/Mdp@//url:14000/bdd
BDD_cible	jdbc:postgresql://url:12000/bdd?user=User&password=Mdp

Les données sélectionnées via les requêtes ou récupérées via les services, sont enregistrées au niveau des « Properties du TestSuite » pour être accessibles depuis tous les TestCase.



A screenshot of the SoapUI TestSuite 1 interface. The window title is 'TestSuite 1'. Below the title bar, there are several icons for running and debugging tests. The main area is divided into sections. The first section is 'TestCases', which contains a green checkmark icon and a list of test cases: 'TestCase : Scénario 1', 'TestCase : Scénario 2', 'TestCase : Clients (disabled)', and 'TestCase : Commandes (disabled)'. Below the test cases, there is a table with two columns: 'Name' and 'Value'. The table contains three rows of data. At the bottom of the interface, there are four tabs: 'Description', 'Properties', 'Setup Script', and 'TearDown Script'. The 'Properties' tab is currently selected. Below the tabs, there is a 'TestSuite Log' section.

Name	Value
Num_Client	23045
Nom_Client	Laporte
Prenom_Client	Charlotte

### Exemples concrets:

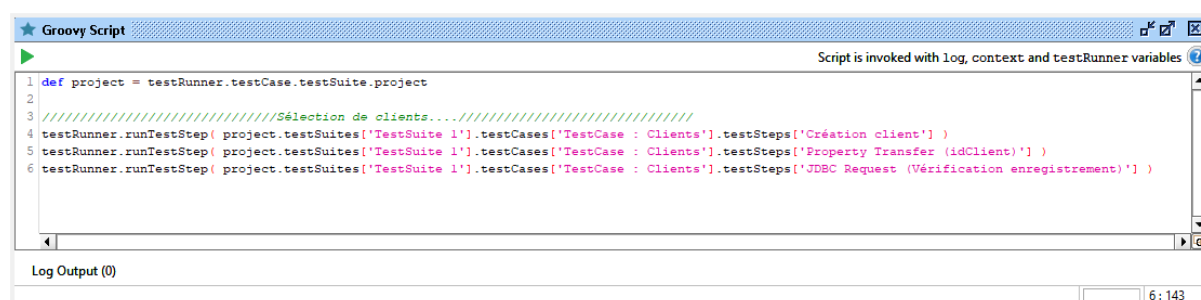
#### a) Automatisation de TNR

Prenons l'exemple de l'automatisation de 2 services REST pour compléter nos TNR.

1. Un service de création de client
2. Un service de recherche de commandes par idClient

Dans notre 1<sup>er</sup> scénario, nous allons mettre en place le script groovy permettant d'appeler :

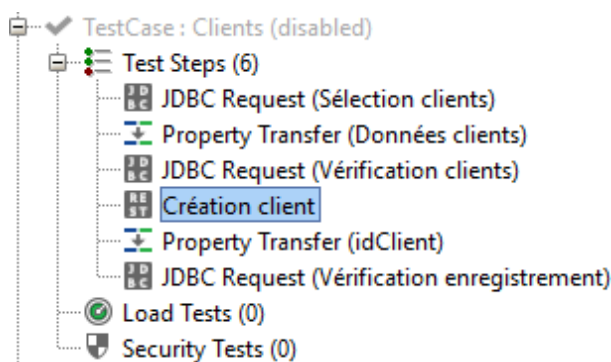
- Le service de création de client
- Le stockage de l'idClient renvoyé par le service
- Une requête en BDD permettant de vérifier l'enregistrement



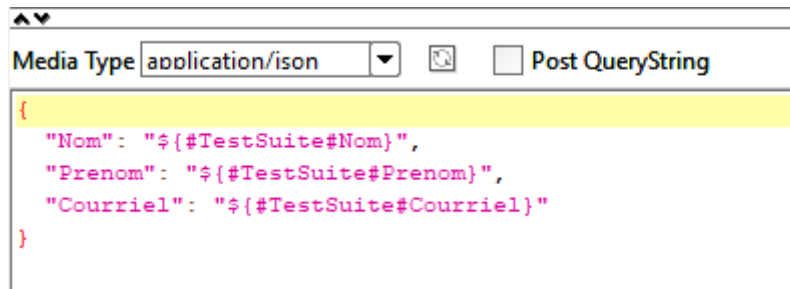
```
1 def project = testRunner.testCase.testSuite.project
2
3 //////////////////////////////////////////////////Sélection de clients.....////////////////////////////////////
4 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['Création client'] )
5 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['Property Transfer (idClient)'] )
6 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Vérification enregistrement)'] )
```

Dans notre brique inactive « Clients », nous allons ajouter :

- Le service de création de client

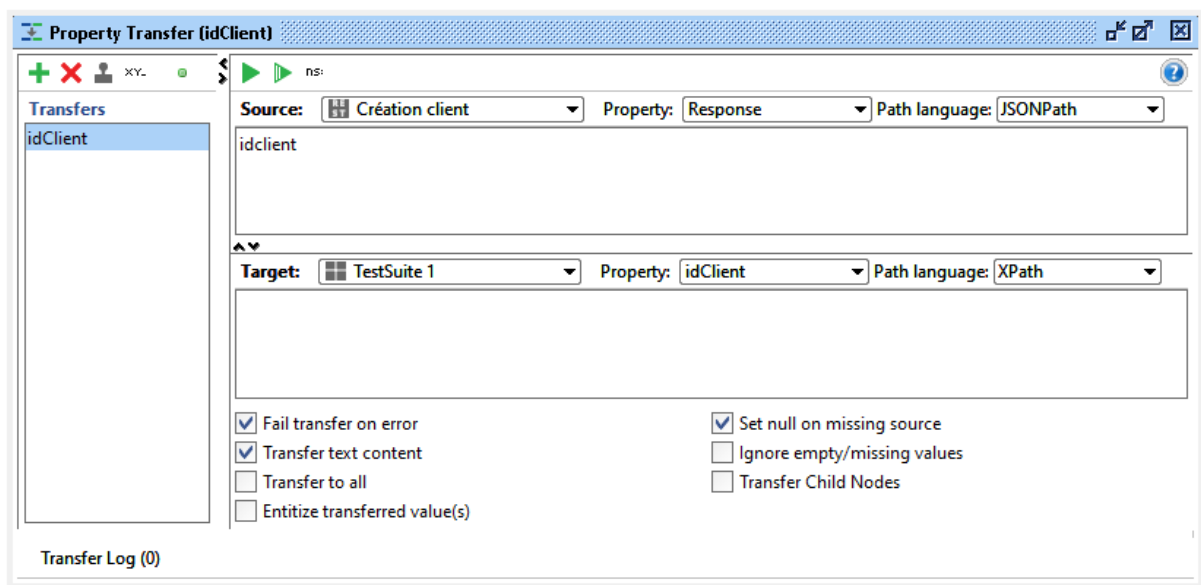


Les données en entrée du service REST sont variabilisées et stockées dans le TestSuite:



```
{
  "Nom": "${#TestSuite#Nom}",
  "Prenom": "${#TestSuite#Prenom}",
  "Courriel": "${#TestSuite#Courriel}"
}
```

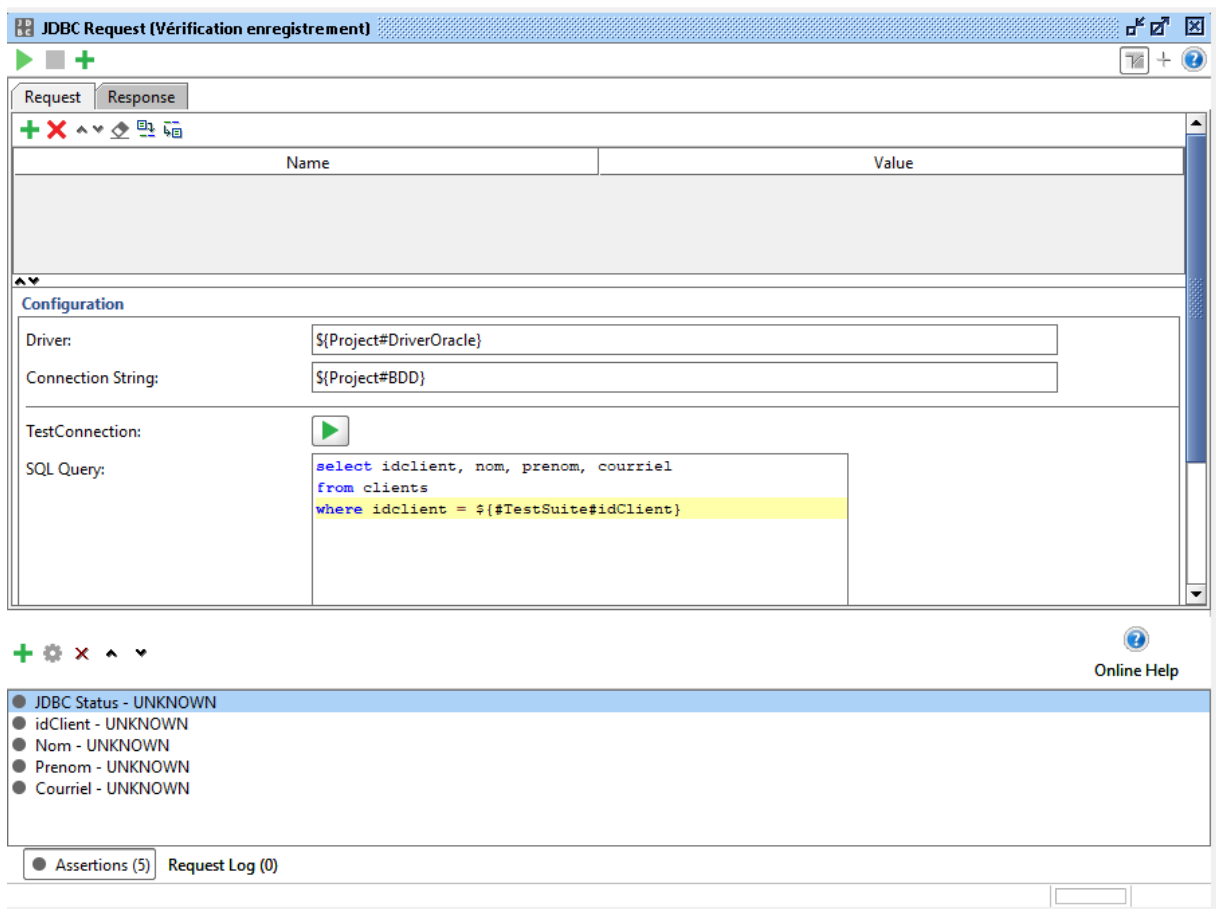
- Un step permettant de stocker l'idClient



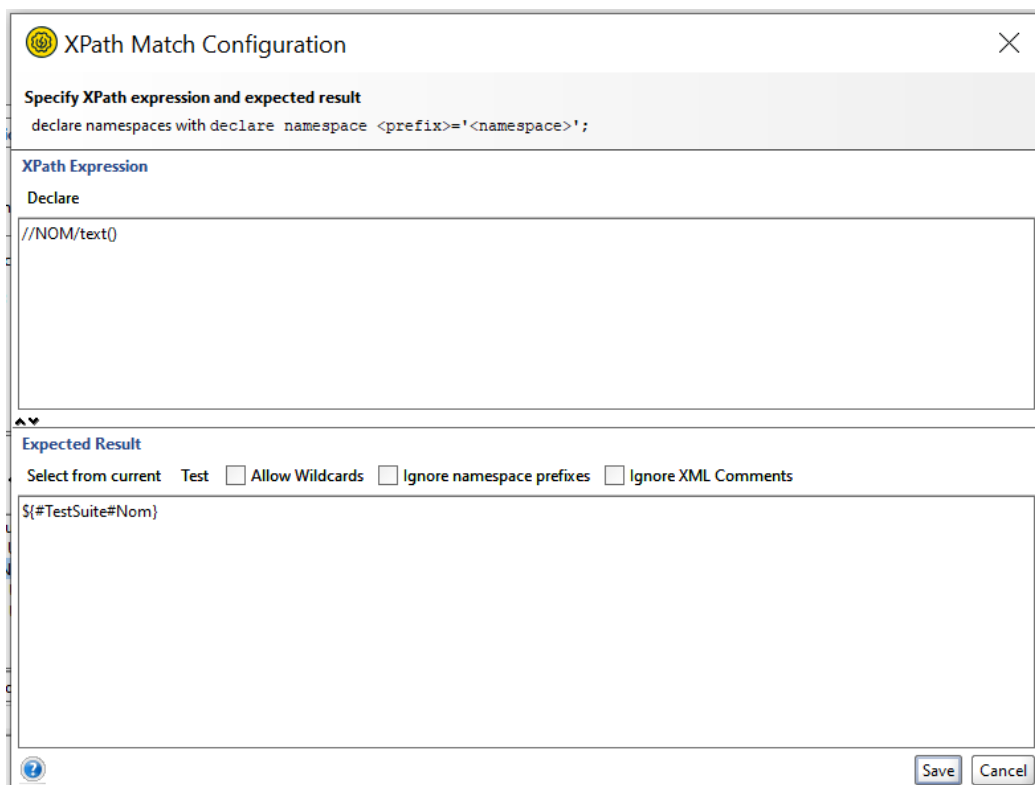
A l'aide de ce step, l'idClient est récupéré dans le retour du service REST et stocké dans le TestSuite.

- La requête permettant de récupérer les informations d'un client

## Automatisation de tests via SoapUI



Chaque information récupérée est vérifiée au sein des assertions, sa valeur doit être identique à celle stockée dans le TestSuite:

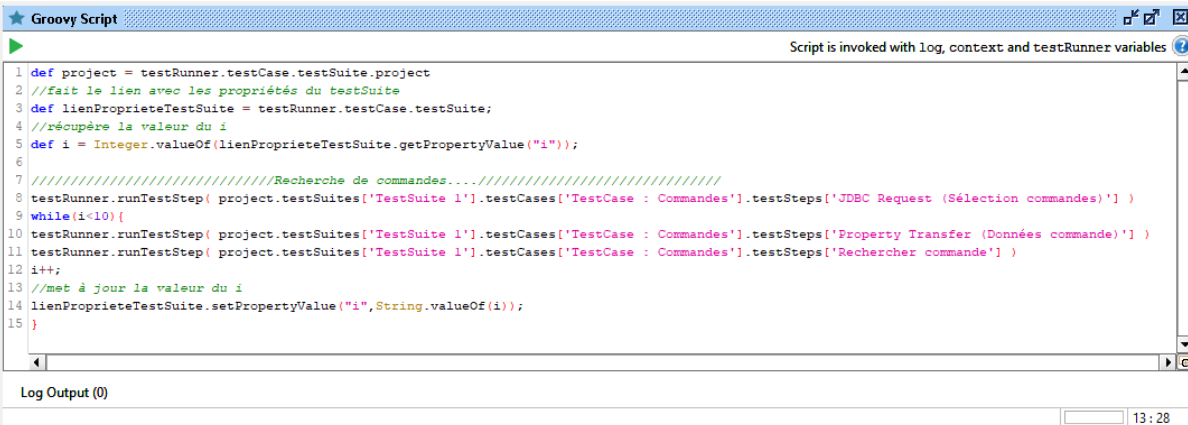




## Automatisation de tests via SoapUI

Dans notre 2<sup>ème</sup> scénario, nous allons mettre en place le script groovy permettant d'appeler:

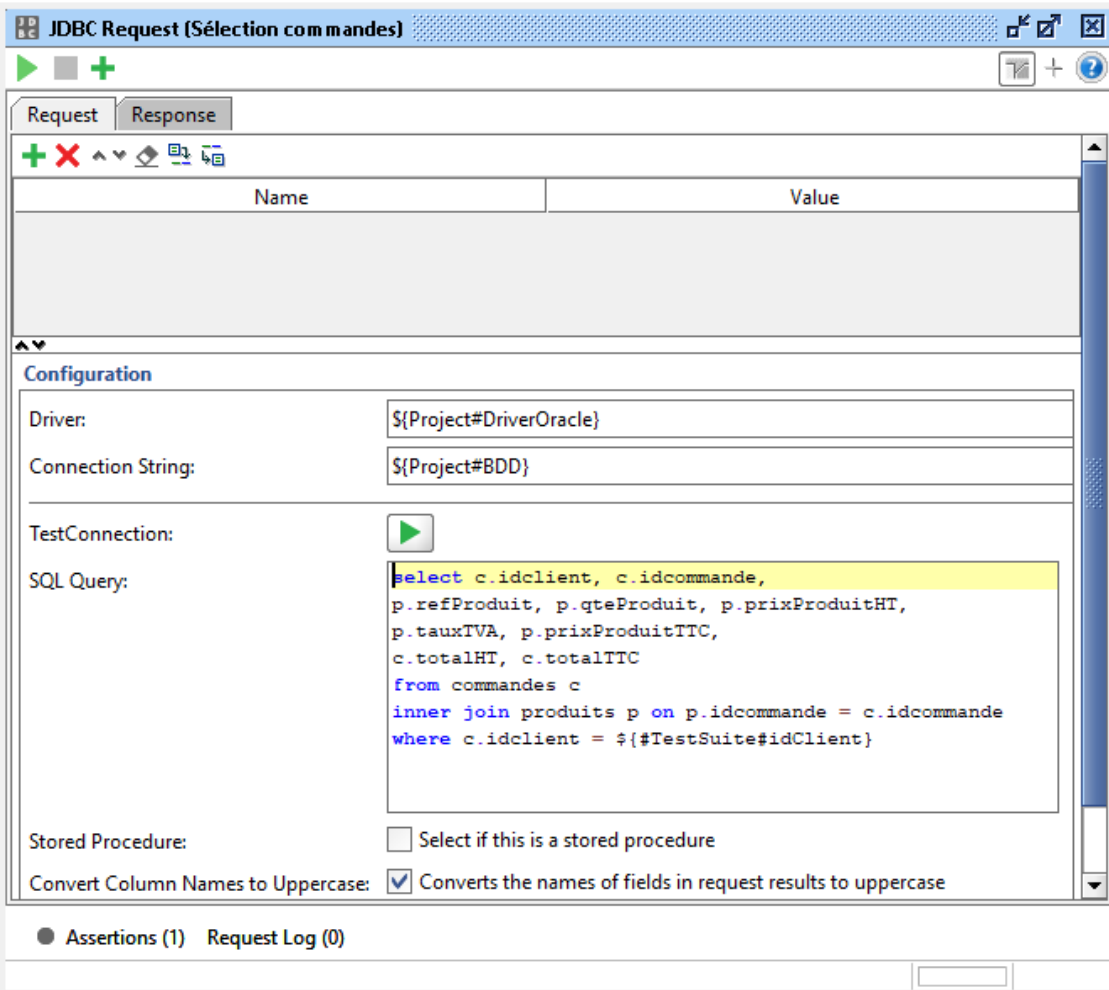
- Une requête en BDD permettant de récupérer les commandes d'un client
- Le stockage des données d'une commande
- Le service de recherche de commandes par idCommande



```
1 def project = testRunner.testCase.testSuite.project
2 //fait le lien avec les propriétés du testSuite
3 def lienProprieteTestSuite = testRunner.testCase.testSuite;
4 //récupère la valeur du i
5 def i = Integer.valueOf(lienProprieteTestSuite.getPropertyValue("i"));
6
7 ////////////////////////////////////////////////////////////////////Recherche de commandes...//////////////////////////////////////////////////////////////////
8 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['JDBC Request (Sélection commandes)'] )
9 while(i<10){
10 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['Property Transfer (Données commande)'] )
11 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['Rechercher commande'] )
12 i++;
13 //met à jour la valeur du i
14 lienProprieteTestSuite.setPropertyValue("i",String.valueOf(i));
15 }
```

Dans notre brique inactive « Commandes », nous allons ajouter:

- La requête permettant de récupérer les commandes d'un client



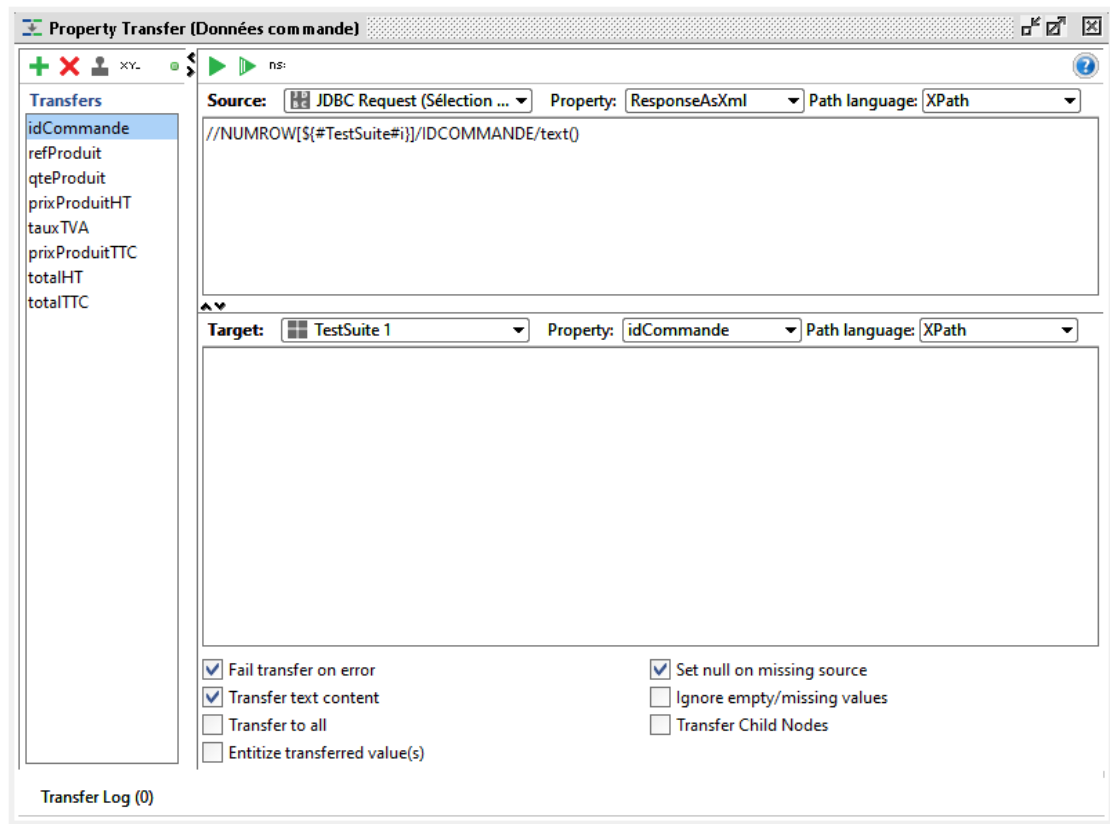
The screenshot shows the configuration for a JDBC Request step. The configuration includes:

- Driver:** \${Project#DriverOracle}
- Connection String:** \${Project#BDD}
- TestConnection:** (indicated by a green play button)
- SQL Query:**

```
select c.idclient, c.idcommande,
p.refProduit, p.qteProduit, p.prixProduitHT,
p.tauxTVA, p.prixProduitTTC,
c.totalHT, c.totalTTC
from commandes c
inner join produits p on p.idcommande = c.idcommande
where c.idclient = ${#TestSuite#idClient}
```
- Stored Procedure:**  Select if this is a stored procedure
- Convert Column Names to Uppercase:**  Converts the names of fields in request results to uppercase

At the bottom, there are indicators for **Assertions (1)** and **Request Log (0)**.

- Un step permettant de stocker les informations d'une commande

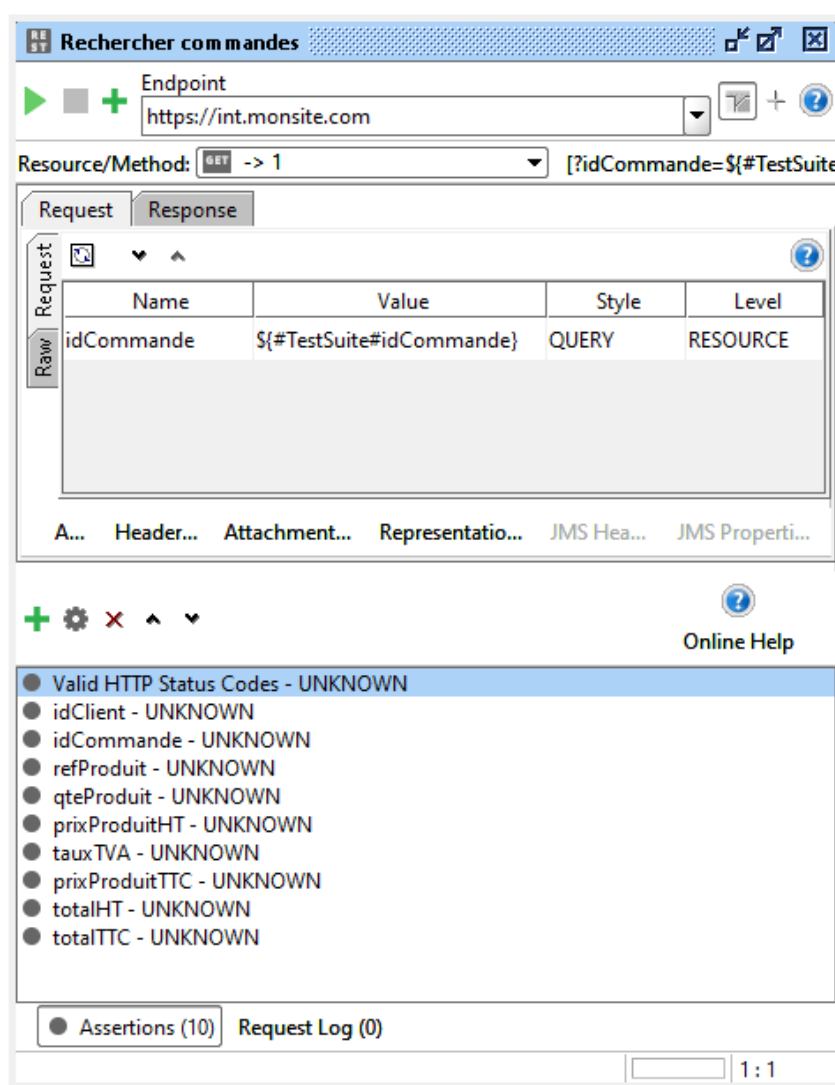


On stocke les données de la 1ère commande pour les comparer à celles de la recherche.

The screenshot shows the 'Custom Properties' tab in SoapUI. It displays a table with two columns: 'Name' and 'Value'. The table contains the following data:

Name	Value
idClient	1463
idCommande	2753
refProduit	A2591
qteProduit	3
prixProduitHT	19,90
tauxTVA	20%
prixProduitTTC	23,88
totalHT	59,70
totalTTC	71,64

- Le service de recherche de commande par idCommande



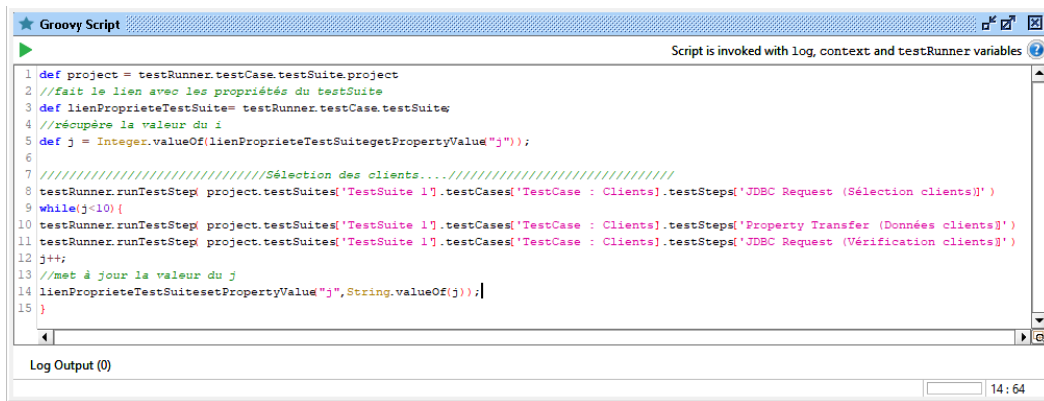
### b) Automatisation des tests d'une migration de BDD

Prenons l'exemple de la migration de notre BDD contenant :

1. La table « Clients »
2. La table « Commandes »

Dans notre 1<sup>er</sup> scénario, nous allons mettre en place le script groovy permettant d'appeler:

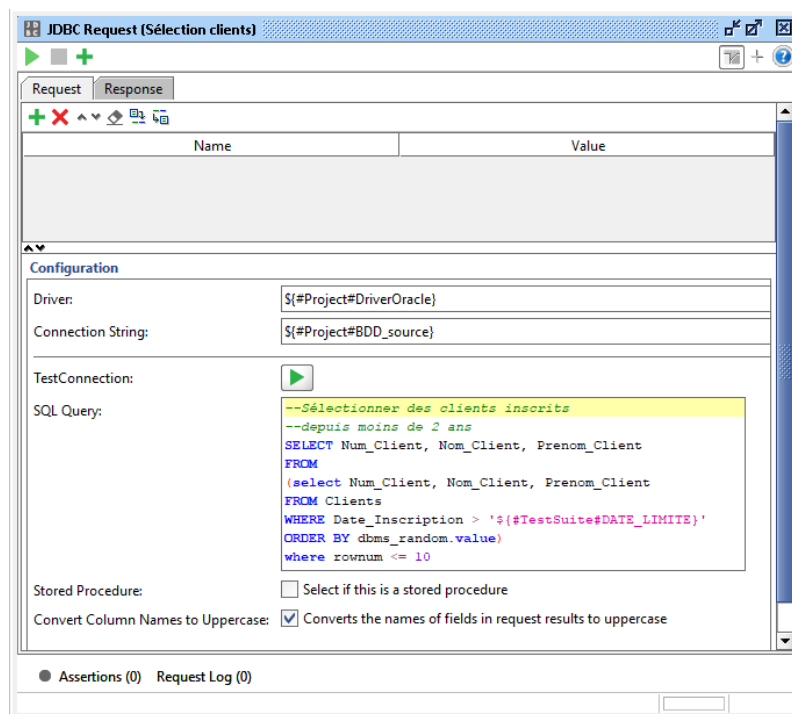
- La requête permettant de sélectionner les clients du périmètre à extraire dans la BDD source
- Le step permettant de stocker les données de chaque client
- La requête permettant de vérifier que les clients sont bien enregistrés dans la BDD cible



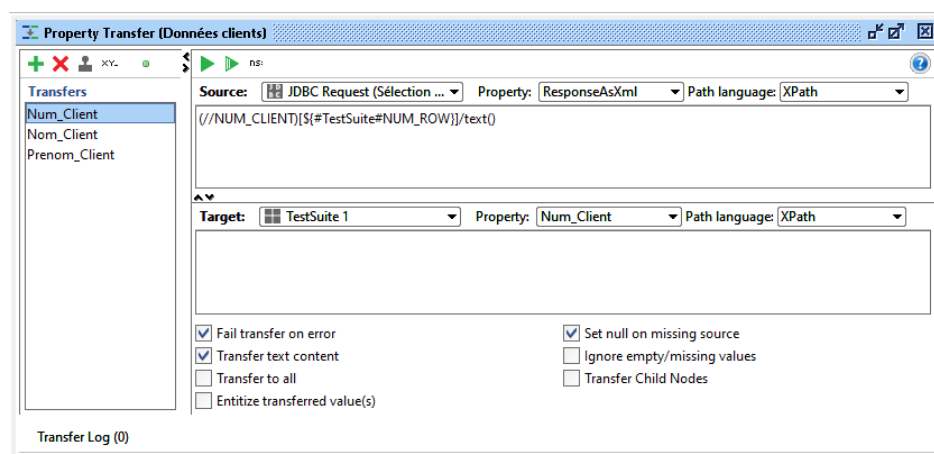
```
1 def project = testRunner.testCase.testSuite.project
2 //fait le lien avec les propriétés du testSuite
3 def lienProprieteTestSuite= testRunner.testCase.testSuite
4 //récupère la valeur du i
5 def j = Integer.valueOf(lienProprieteTestSuite.getPropertyValue("j"));
6
7 //////////////////////////////////////////////////Sélection des clients.....////////////////////////////////////
8 testRunner.runTestStep project.testSuites["TestSuite 1"].testCases["TestCase : Clients"].testSteps["JDBC Request (Sélection clients)"]
9 while(j<10){
10 testRunner.runTestStep project.testSuites["TestSuite 1"].testCases["TestCase : Clients"].testSteps["Property Transfer (Données clients)"]
11 testRunner.runTestStep project.testSuites["TestSuite 1"].testCases["TestCase : Clients"].testSteps["JDBC Request (Vérification clients)"]
12 j++;
13 //met à jour la valeur du j
14 lienProprieteTestSuite.setPropertyValue("j",String.valueOf(j));
15 }
```

Dans notre brigue inactive « Clients », nous allons ajouter:

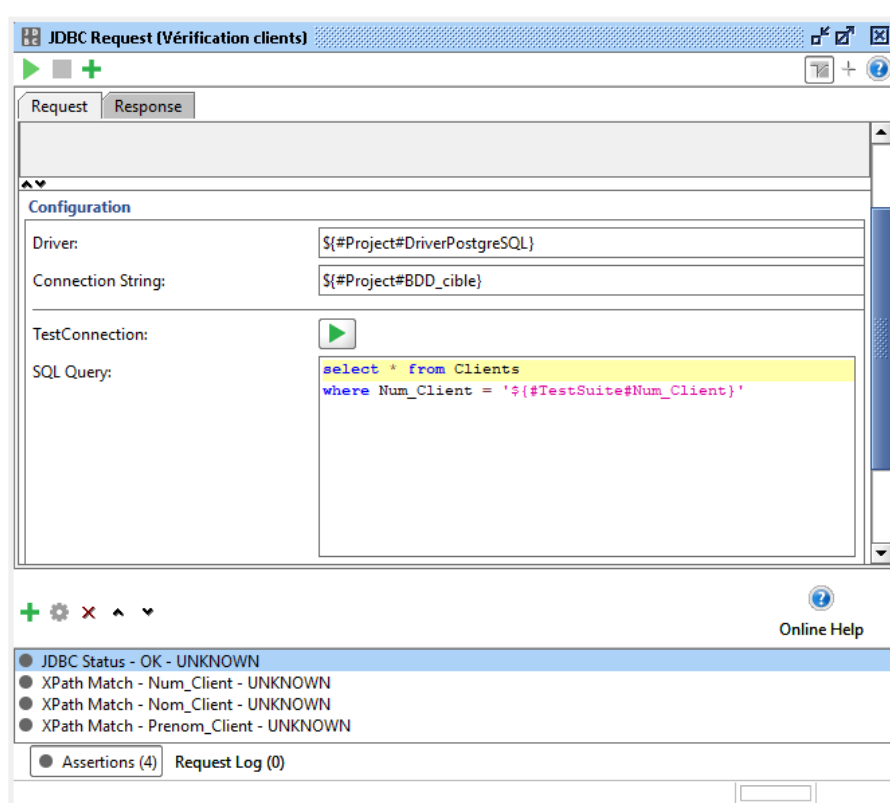
- La requête permettant de sélectionner les clients dans la BDD source



- Le step permettant de stocker les données de chaque client



- La requête permettant de vérifier l'enregistrement des clients dans la BDD cible



Pour compléter ce scénario, nous pouvons ajouter une requête permettant de vérifier qu'aucun client hors périmètre n'est présent dans la BDD cible.

Dans notre 2<sup>ème</sup> scénario, nous allons mettre en place le script groovy permettant d'appeler:

- La requête permettant de sélectionner les commandes des clients extraits et datant de moins de 6 mois
- Le step permettant de stocker les données de chaque commande
- La requête permettant de vérifier que les commandes sont bien enregistrées dans la BDD cible

## Automatisation de tests via SoapUI

```
1 def project = testRunner.testCase.testSuite.project
2 //fait le lien avec les propriétés du testSuite
3 def lienProprieteTestSuite = testRunner.testCase.testSuite;
4 //récupère la valeur du k
5 def k = Integer.valueOf(lienProprieteTestSuite.getPropertyValue("k"));
6 ////////////////Sélection des commandes clients...////////////////////
7 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Sélection clients)'] )
8 while(k<10) {
9 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['Property Transfer (Données clients)'] )
10 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['JDBC Request (Sélection commandes)'] )
11 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['Property Transfer (Données commandes)'] )
12 testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Commandes'].testSteps['JDBC Request (Vérification commandes)'] )
13 k++;
14 //met à jour k
15 lienPropriete.setPropertyValue("k",String.valueOf(k));
16 }
```

Dans notre bricque « Commandes », nous allons ajouter:

- La requête permettant de sélectionner les commandes dans la BDD source

Request Response

Name	Value
------	-------

Configuration

Driver:

Connection String:

TestConnection:

SQL Query: 

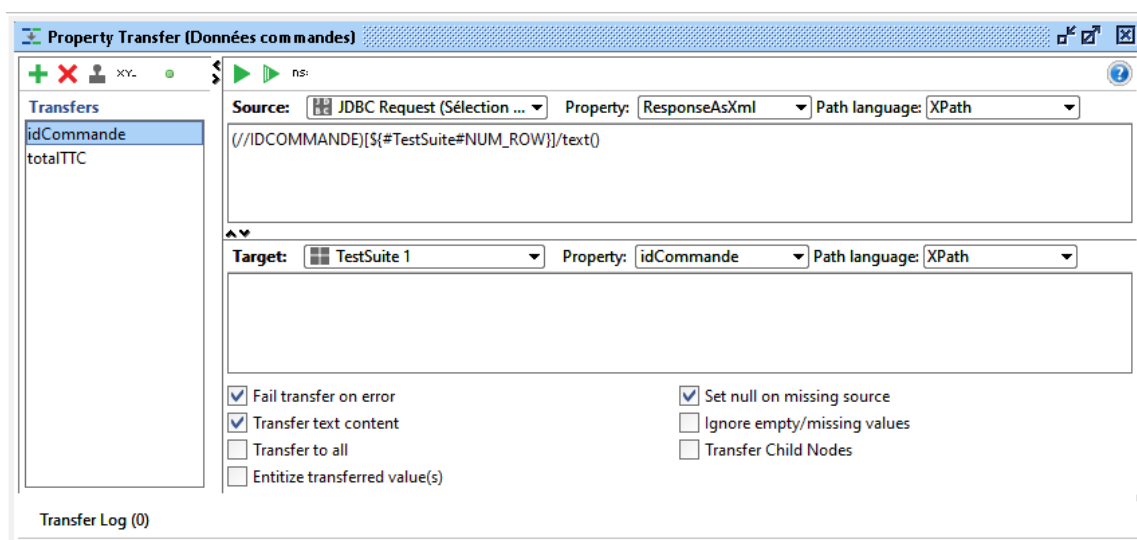
```
--Sélectionner les commandes des clients
--datant de moins de 6 mois
SELECT cl.Num_Client, c.idCommande, c.totalTTC
FROM Commandes c
INNER JOIN Clients cl on c.idclient = cl.Num_Client
WHERE c.dateAchat > '${TestSuite#DATE_COMMANDE_LIMITE}'
AND c.idclient = '${TestSuite#Num_Client}'
```

Stored Procedure:  Select if this is a stored procedure

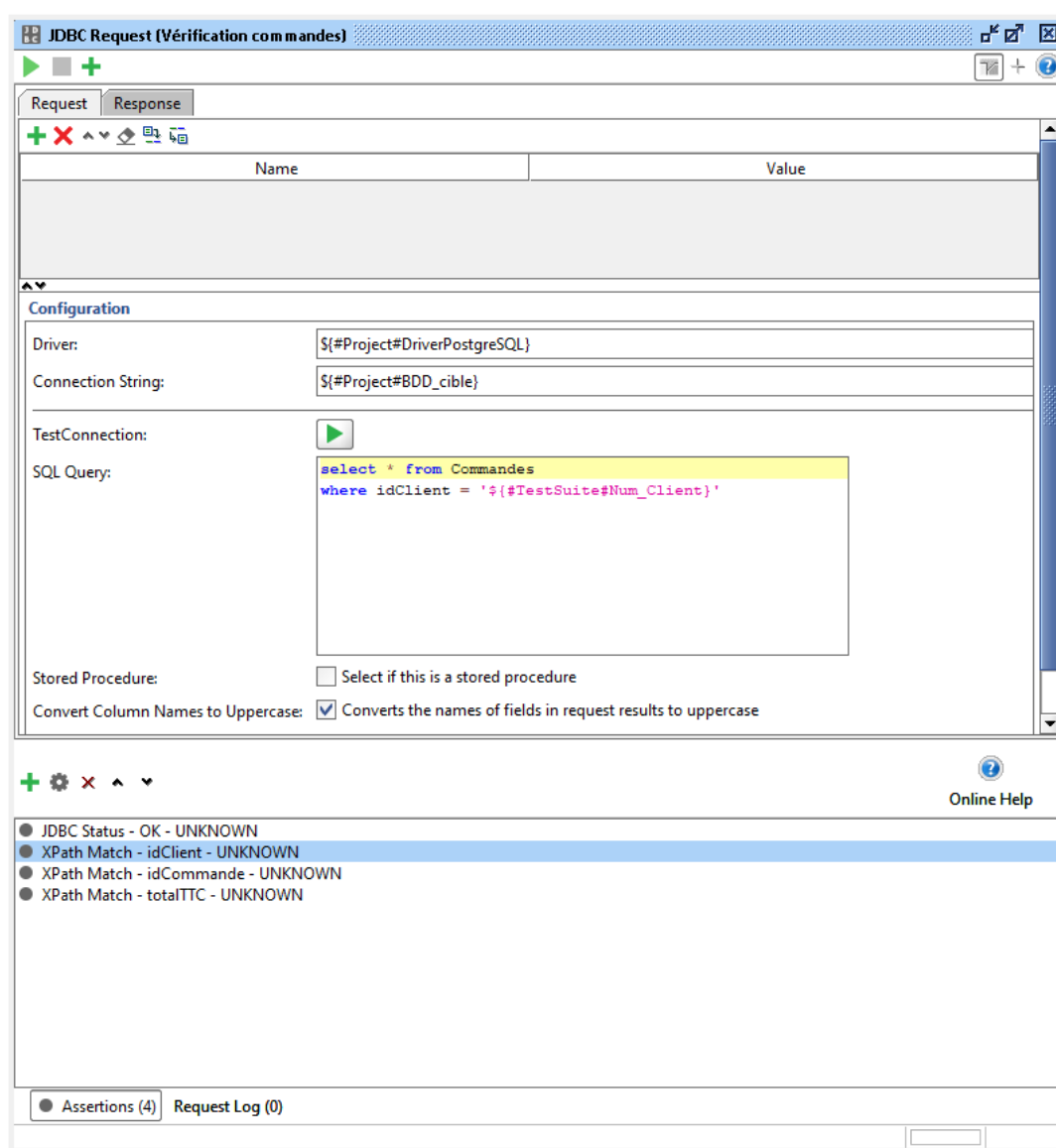
Convert Column Names to Uppercase:  Converts the names of fields in request results to uppercase

● Assertions (0) Request Log (0)

- Le step permettant de stocker les données de chaque commande



- La requête permettant de vérifier l'enregistrement des commandes dans la BDD cible



Pour compléter ce scénario, nous pouvons ajouter une requête permettant de vérifier qu'aucune commande datant de plus de 6 mois ou n'appartenant pas à un client du périmètre n'est présente dans la BDD cible.

En fonction de la livraison des différents travaux, si les extractions des tables « Clients » et « Commandes » sont livrées au même moment, un seul scénario peut être envisagé.

- Sélectionner les clients de la BDD source
- Stocker les données du 1er client
- Sélectionner les commandes du 1<sup>er</sup> client
- Stocker les données des commandes du 1<sup>er</sup> client
- Vérifier l'enregistrement des données du 1<sup>er</sup> client



## Automatisation de tests via SoapUI

- Vérifier l'enregistrement des données des commandes du 1<sup>er</sup> client
- Boucler pour vérifier l'ensemble des extractions clients/commandes

**Amis testeurs, lancez-vous dans l'automatisation avec SoapUI !**