



Module 3 Développer la persistance des données

Implémenter la DB	Séance	S04	Activité	A-005
-------------------	--------	-----	----------	-------

L'activité proposée doit vous permettre de vous initier aux techniques de programmation de fonctions SQL avec des langages de haut niveau du fait des possibilités offertes par l'implémentation du framework .net au niveau de SQL Server.

Sommaire de l'activité proposée :

1	Vérification de règles complexes	2
2	Implémenter des fonctions complexes avec intégration CLR	2
2.1	Définir les fonctions	2
2.2	Lier les fonctions externes	2
2.3	Création des fonctions	3
2.4	Lier la bibliothèque de fonctions.....	4
2.5	Définir la fonction SQL.....	5
2.6	Tester la fonction SQL.....	5
2.7	Mettre à jour l'assembly	6
3	Implémentation au sein de votre projet	6

1 Vérification de règles complexes

Nous pouvons aussi étendre les possibilités de SQL server grâce à l'intégration du runtime d'exécution .net dans SQL Server.

Nous pouvons ainsi programmer des fonctions en langage de haut niveau et invoquer ces fonctions au niveau de SQL Server. Plus beaucoup de limites donc

Pour illustrer ces possibilités, nous allons implémenter :

- une méthode qui permet de vérifier que la valeur d'une colonne respecte le patron d'une expression régulière. Nous utiliserons cette méthode pour vérifier :
 - Le matricule stagiaire (8 chiffres)
 - Le matricule formateur (nnAAnnn ou n est un chiffre et A un caractère alphabétique)
 - L'adresse mail de la personne
- Une méthode pour implémenter le contrôle du numéro de SIRET. C'est l'algorithme de luhn qui est utilisé pour vérifier la validité du numéro de SIRET. La formule de luhn est utilisée dans le calcul de nombreuses clés de contrôles. Voir article sur Wikipedia https://fr.wikipedia.org/wiki/Formule_de_Luhn. J'ai codé la fonction pour vous :

```
internal static bool IsLuhnValide(string valeur)
{
    int chiffre;
    int somme = 0;
    int impair = valeur.Length & 1;

    for (int i = 0; i < valeur.Length; i++)
    {
        chiffre = int.Parse(valeur[i].ToString()) * (2 - (i+impair) % 2);
        somme += chiffre > 9 ? chiffre - 9 : chiffre;
    }
    return (somme % 10 == 0);
}
```

2 Implémenter des fonctions complexes avec intégration CLR

Nous allons voir ici comment tirer avantages de l'implémentation du CLR (Common Langage Runtime) dans SQL Server.

Nous allons définir des fonctions en C# qui seront invoquées depuis SQL Server pour vérifier qu'une information respecte bien un modèle fourni sous la forme d'une expression régulière.

2.1 Définir les fonctions

Les fonctions sont développées et fournies sous la forme d'une bibliothèque de fonctions (dll).

Quelques règles doivent être respectées en particulier sur la définition des paramètres qui sont fournis sous la forme de paramètres SQL.

2.2 Lier les fonctions externes

L'assemblage de type bibliothèque de fonctions (dll) issu de la compilation du projet de classes doit être enregistré au niveau de SQL (register).

Une fonction SQL sera ensuite liée à la fonction externe.

2.3 Création des fonctions

Créer un projet de type bibliothèque de classes.

Les fonctions devront être définies comme **statiques** et **publiques**.

Il est possible de compléter leur définition afin d'obtenir de meilleures performances.

Ici, par exemple, la fonction est définie comme déterministe. C'est le cas de la plupart des fonctions. Ce terme indique que la fonction renvoie toujours la même valeur si les mêmes arguments lui sont fournis en entrée. *Il existe des fonctions non déterministes comme par exemple celle qui fournirait une série de valeurs aléatoires avec initialisation spécifique interne à partir d'une donnée d'horodatage.*

Les types définis en argument sont des types propres à SQL : cela est préférable même si ce n'est pas une obligation. Obligation toutefois d'utiliser des types compatibles.

```
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.Text.RegularExpressions;

namespace TypesSQL
{
    1 référence
    public static class ExpressionsRegulieres
    {
        [SqlFunctionAttribute(IsDeterministic=true)]
        1 référence
        public static SqlBoolean ExpressionRegIsMatch(SqlString valeur, SqlString motif)
        {
            if (valeur.IsNull || motif.IsNull)
            {
                return SqlBoolean.False;
            }
            return Regex.IsMatch(valeur.Value, motif.Value);
        }
    }
}
```

Vous remarquerez la présence des espaces de noms `SqlTypes` et `SqlServer.Server` pour accès respectivement aux types et aux annotations.

Vous devez ensuite compiler ce projet en vous assurant de **définir une version du CLR qui soit implémentée au niveau de SQL Server**. Pour des questions de performances, nous utiliserons une version de production de la bibliothèque de fonctions.

Définition des options de génération au niveau du projet :

The screenshot shows the 'Application' tab of the project properties for 'TypesSQL'. The 'Configuration' and 'Platform' are both set to 'Non applicable'. The 'Nom de l'assembly' is 'TypesSQL'. The 'Espace de noms par défaut' is 'TypesSQL'. The 'Framework cible' is '.NET Framework 3.5'. The 'Type de sortie' is 'Bibliothèque de classes'. The 'Objet de démarrage' is '(Non défini)'. There is a button 'Informations de l'assembly...'.

Application	Configuration : (Release) active	Plateforme : (Any CPU) active
Générer		
Événements de build	Général	

J'ai utilisé comme cible le framework 3.5 pour implémenter ma bibliothèque sous SQL Server 2008

2.4 Lier la bibliothèque de fonctions

Il faut tout d'abord s'assurer que les paramètres du serveur autorisent l'utilisation du CLR.

La procédure stockée système SP_Configure permet de modifier ces paramètres. Il faut ensuite exécuter l'instruction RECONFIGURE pour que les nouveaux paramètres soient pris en compte.

La dll sera définie globalement sur le serveur mais liée à une base de données. Ici, si l'assemblage existe déjà, il est supprimé. Vous pouvez lui affecter un nom sans rapport direct avec celui de la dll.

A noter : toutes les références à l'assemblage doivent être supprimées avant la suppression de l'assemblage.

```
-- Autoriser l'intégration de fonctions et types CLR

sp_configure 'clr enabled', 1
GO
RECONFIGURE
GO

-- Définition de la base de donnée cible
USE AFPA2015

-- Suppression de l'assemblage
IF EXISTS (SELECT * FROM sys.assemblies asms
WHERE asms.name = N'IntegrationSQL' and is_user_defined = 1)
DROP ASSEMBLY [IntegrationSQL]
GO
```

L'option suivante, et la plus importante, est la création de la référence à l'assemblage.

```
-- Inscrire la dll

CREATE ASSEMBLY
IntegrationSQL -- Nom SQL de l'assemblage

FROM 'D:\ImplementationSQL\TypesSQL\bin\Release\TypesSQL.dll'
-- nom et chemin de la DLL
WITH PERMISSION_SET = SAFE -- recommandé voir infos
GO
```

Il est recommandé d'utiliser SAFE au niveau du paramètre Permission_Set.

Il s'agit de l'ensemble d'autorisations le plus restrictif. Le code exécuté par un assembly avec les autorisations SAFE ne peut pas accéder aux ressources système externes telles que les fichiers, le réseau, les variables d'environnement ou le Registre. Voir documentation SQL Server.

2.5 Définir la fonction SQL

Dernière étape, la définition de la fonction SQL

```
-- Respecter la signature de la méthode.  
-- Vous pouvez utiliser des noms différents  
CREATE FUNCTION VerifierExpReg(@texte nvarchar(max), @modele nvarchar(255))  
RETURNS BIT  
-- Syntaxe Assemblage.[EspaceNoms.Type].Fonction  
AS EXTERNAL NAME [IntegrationSQL].[TypesSQL.ExpressionsRegulieres].ExpressionRegIsMatch
```

Respectez bien la syntaxe pour référencer la fonction externe :

Assemblage.[EspaceNoms.Type].NomFonction

2.6 Tester la fonction SQL

Réalisez un script pour tests des expressions régulières.

```
Select 'Attendu 1', dbo.VerifierExpReg('ab', 'a')  
Select 'Attendu 0', dbo.VerifierExpReg('ab', 'c')  
  
Select 'Attendu 1', dbo.VerifierExpReg('01346', '^ [0-9] {5}$')  
Select 'Attendu 0', dbo.VerifierExpReg('0A346', '^ [0-9] {5}$')  
Select 'Attendu 0', dbo.VerifierExpReg('0134', '^ [0-9] {5}$')  
|  
Select 'Attendu 1', dbo.VerifierExpReg('01345', '^ \d{5}$')  
Select 'Attendu 0', dbo.VerifierExpReg('0134', '^ \d{5}$')
```

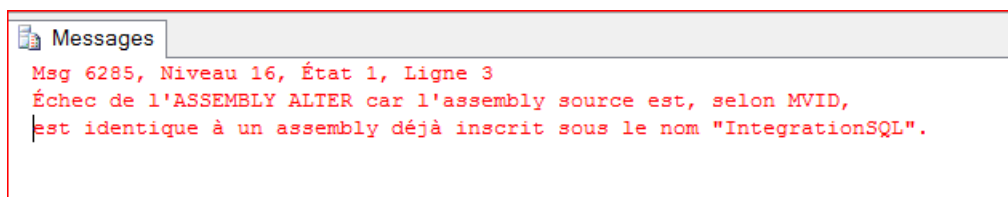
Il peut être utile de développer en C# un projet de test qui facilitera le cas échéant les tests de mise au point...

2.7 Mettre à jour l'assemblage

Pour mettre à jour l'assemblage utilisez la commande ALTER. La nouvelle version de la DLL sera alors liée à SQL Server.

```
-- Modification de l'assemblage
ALTER ASSEMBLY [IntegrationSQL]
FROM 'D:\ImplementationSQL\TypesSQL.dll'
WITH PERMISSION_SET = SAFE
GO
```

Vous obtiendrez une erreur si le numéro de version n'a pas été changé comme ci-dessous le système effectuant un test de la version : MVID correspond à l'Identifiant du numéro de version complet de l'assemblage.



The screenshot shows a 'Messages' window with the following text:

```
Msg 6285, Niveau 16, État 1, Ligne 3
Échec de l'ASSEMBLY ALTER car l'assembly source est, selon MVID,
est identique à un assembly déjà inscrit sous le nom "IntegrationSQL".
```

3 Implémentation au sein de votre projet

Nous pouvons maintenant programmer des contraintes de validité complexe au sein de nos triggers.

Vous devrez modifier la méthode de vérification par expressions régulières en gérant le paramètre options pour être conforme à la norme. Voir l'article *Options des expressions régulières* sur msdn.

Implémenter au niveau des transactions sur la table Beneficiaire la vérification :

- de la structure du matricule
- de la structure de l'adresse mail lorsqu'elle est fournie.