Forums Tutoriels Magazine FAQs Blogs Projets Chat Newsletter Études Emploi Club Contacts GO !

Developpez.com
Club des professionnels de l'informatique

Developpez.com
Plus de 1000 forums
et jusqu'à 5000 nouveaux messages par jour

Accueil Conception Java .NET Dév. Web EDI Langages SGBD Office Solutions d'entreprise Applications Systèmes

SGBD 4D Access DB2 Firebird InterBase MySQL Oracle PostgreSQL SQL-Server Sybase

ACCUEIL SQL-SERVER FORUM SQL-SERVER F.A.Q SQL-SERVER TUTORIELS SQL-SERVER SOURCES SQL-SERVER

LIVRES SQL-SERVER

Procédures stockées SQL Server 2005 en .NET

Date de publication : 04/09/2006

Par rudib (Rudi Bruchez)

Article paru dans le magazine Programmez! de février 2006. Microsoft SQL server 2005 apporte de nouvelles fonctionnalités majeures au produit, dont une des plus importante est l'intégration du framework .NET dans le moteur. Cet article va vous guider dans la création de code .NET destiné à être embarqué dans SQL Server.

I. Utilisation et Performances
I-A. Sécurité
II. Développer une procédure stockées avec un langage de la CLR
II-A. Création de la Procédure
II-B. Connexion de Contexte
II-C. Objet de Contexte
II-D. Exemple de Procédure
II-E. Quelques Notes
II-F. Transactions
III. Déploiement

I. Utilisation et Performances

Utilisez la technologie à bon escient. Le langage de requêtes natif de SQL Server, Transact SQL, doit rester la solution de choix pour toute requête, et il faut autant que possible essayer d'exprimer son code en T-SQL. Jamais du code .NET ne sera plus efficace que T-SQL pour parler au moteur relationnel. T-SQL est ce qu'on appelle un langage assertionnel, qui permet d'exprimer la requête dans une optique relationnelle et ensembliste, laissant le soin au moteur de SQL Server lui-même de construire un plan de requête optimisé. Les langages .NET sont des langages procéduraux, qui pour atteindre le même résultat, nécessitent de construire manuellement cette logique. SQL s'adresse à des ensembles de données, alors que les langages .NET ne peuvent traiter que des éléments.

Par contre SQL est très peu souple en ce qui concerne les traitements en boucle et l'algorithmie. Exprimer du code procédural est en général plus lent en T-SQL que dans un langage compilé. En résumé, dans des cas comme la manipulation de chaînes de caractères ou des opérations mathématiques, l'intégration du CLR peut s'avouer un allié précieux.

I-A. Sécurité

Un langage MSIL .NET est un langage complet permettant de créer du code aux possibilités infinies, notamment il permet ce qui était impossible au pur langage SQL (sauf en passant par quelques procédures stockées étendues dont l'exécution était maîtrisable par l'administrateur du système), comme de gérer des entrées/sorties. Cela pose évidemment la question de la sécurité. Nous avons vu depuis quelques années quels problèmes peuvent déjà générer de simples langages de macro intégrés aux outils bureautiques. Un serveur de bases de données est souvent un élément stratégique de l'informatique de l'entreprise, il est donc crucial de maîtriser l'exécution du code dans son périmètre.

L'introduction du framework .NET dans SQL Server a demandé à Microsoft une réécriture et une solidification du framework, dont les modifications de la version 2.0 sont en partie dues aux exigences de l'équipe de développement de SQL Server. Le code .NET qui s'exécute dans le contexte de SQL Server l'est dans un système d'exploitation virtuel, nommé la couche SQLOS, une forme de système d'exploitation en mode utilisateur. Cette couche limite donc fortement les possibilités de code dangereux s'exécutant dans SQL Server. Le CLR qui est lancé par SQL Server est appelé en interne SQLCLR.

Lorsque vous ajoutez un assemblage à l'intérieur de SQL Server, vous pouvez indiquer explicitement son niveau d'autorisation. Par défaut, le niveau est en mode *safe,* ce qui est logique pour des utilisations en base de données: vous n'avez normalement pas besoin d'accéder au monde extérieur. Pour indiquer un niveau différent, vous pouvez utiliser l'extension WITH PERMISSION_SET, à la commande CREATE ASSEMBLY, ou spécifier le *permission level* dans les propriétés du projet Visual Studio (onglet *database*).

II. Développer une procédure stockées avec un

langage de la CLR

La première chose dont vous devez vous assurer, est que le support du CLR est activé dans SQL Server. Dans SQL Server 2005, les fonctionnalités délicates en termes de sécurité sont désactivées par défaut lors de l'installation.

Pour cela, vous devez poser l'option *clr enabled* à 1, cela peut se faire avec la commande suivante :

```
sp_configure 'clr enabled', 1
```

Vous pouvez l'activer visuellement à travers l'outil « Surface Area Configuration », qui permet de gérer les fonctionnalités activées ou désactivées. Sous CLR Integration, cochez « Enable CLR Integration ».

Une chose est à préciser : le CLR n'est pas lancé au chargement de SQL Server, mais bien la première fois qu'un assemblage est exécuté dans le contexte de SQL Server.

II-A. Création de la Procédure

Nous allons brièvement détailler la façon de créer une procédure stockée en C#, en utilisant Visual Studio 2005.

Dans Visual Studio, Créez un nouveau projet C#, en Sélectionnant *Database*, *SQL Server Project*. Une question vous est posée sur l'activation de la fonctionnalité de déboguage du code managé. Il faut savoir que le code que vous allez développer sera exécuté à l'intérieur du process de SQL Server. Si vous lancez la session de déboguage à partir de Visual Studio, indiquez le code SQL qui lancera votre procédure dans le fichier test.sql automatiquement ajouté au projet. Visual Studio s'occupera de l'attachement au processus SQL Server. Si vous voulez lancer manuellement l'exécution de la procédure, posez un point d'arrêt dans votre code, attachez le au processus SQL Server (sqlservr.exe) à l'aide de la commande *Attach to Process* du menu *Debug*, et exécutez votre procédure depuis Management Studio par exemple. Après l'exécution, vérifiez que le processus est détaché, sinon faites un *detach all* pour rendre son autonomie à SQL Server.

Lorsque le projet est créé, sélectionnez dans le menu contextuel du projet, dans l'explorateur de solutions, *Add -> Stored Procedure*. Visual Sudio 2005 génère un squelette de code, utilisant les bibliothèques SQL Server, et créant une méthode dans la classe *StoredProcedures*. La méthode doit être publique et statique. La classe *StoredProcedures* est déclarée *partial*. Partial est un nouveau mot-clé du framework 2.0 qui permet de disperser le code d'une classe dans plusieurs fichiers, le soin de regrouper le code complet de la classe revenant au compilateur. Cet élément est optionnel. Ci-dessous, le code généré par Visual Sudio :

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;


public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void StoredProcedure1()
    {
        // Put your code here
    }
};
```

Les namespaces déclarés sont:

System.Data.SqlClient	**Le client d'accès aux données ADO.NET.**
System.Data.SqlTypes	Les types de données SQL
Microsoft.SqlServer.Server	Donne accès au contexte de SQL Server

Vous voyez que le client ADO.NET est déclaré. En effet, pour accéder à vos données dans ce contexte, vous devez nécessairement repasser par ADO.NET. Notez que depuis ce contexte vous pouvez très bien accéder à d'autres sources de données, par exemple des providers OleDb en déclarant System.Data.OleDb. N'oubliez pas que pour permettre l'accès à une autre source de données que le serveur dans lequel s'exécute la procédure stockée, le PERMISSION_SET doit être posé à EXTERNAL_ACCESS.

[Microsoft.SqlServer.Server.SqlProcedure] situé devant la déclaration de la méthode est un attribut qui sera utilisé par Visual Studio pour le déploiement du code.

L'espace de nom System.Data.SqlTypes contient la définition des données natives du serveur SQL. Pour des raisons de cohérence et d'efficacité, utilisez ces types de données, qui sont préfixés par Sql. Vous éviterez ainsi toute conversion implicite entre des types de donnée .NET et des types SQL, et rendrez votre code plus rapide. Vous pourrez aussi gérer la nullabilité à l'aide de ces types.

II-B. Connexion de Contexte

Afin de récupérer la connexion dans laquelle votre code s'exécute, vous avez à votre disposition une chaîne de connexion dite connexion de contexte, dont la syntaxe est :

```
SqlConnection c = new SqlConnection("context connection=true");
```

La connexion de contexte est très rapide, car elle appelle le serveur directement, sans passer par le protocole de réseau que les connexions habituelles utilisent.

II-C. Objet de Contexte

A l'intérieur de votre code, le namespace Microsoft.SqlServer.Server vous donne accès à l'object de contexte: SqlContext. Celui-ci vous permet notamment d'accéder à la connexion avec le client, par l'intermédiaire de l'objet SqlPipe :

```
SqlContext.Pipe.Send("quelqu'un m'appelle ?");
```

La méthode Send de Pipe permet d'envoyer au client connecté soit une chaîne de caractères, soit un enregistrement (SqlDataRecord), soit un jeu d'enregistrements (SqlDataReader).

La propriété IsAvailable de l'objet Pipe vous indique si la connexion de contexte est disponible, donc si le code s'exécute dans SQLCLR.

II-D. Exemple de Procédure

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void GetSessions(SqlString user)
    {
        try
        {
            using (SqlConnection cn = new SqlConnection("context connection=true"))
            {
                if (user.IsNull)
                {
                    user = SqlContext.WindowsIdentity.Name;
                }
                cn.Open();
                string sql = @" SELECT *
                                FROM sys.dm_exec_sessions
                                WHERE login_name = @system_user";
                SqlCommand cmd = new SqlCommand(sql, cn);
                cmd.Parameters.AddWithValue("@system_user", user);
                SqlContext.Pipe.Send(cmd.ExecuteReader());
            }
        }
        catch (Exception e)
        {
            SqlPipe pipe = SqlContext.Pipe;
            pipe.Send("erreur d'execution");
            pipe.Send(e.Message);
            pipe.Send(e.StackTrace);
        }
    }
};
```

L'exemple suivant est une procédure parfaitement inutile, dont l'existence est dédiée à la démonstration, elle pourrait parfaitement être écrite en T-SQL, mais vous donne une idée de base de l'écriture de procédures en C#.

Afin de vous assurer de toujours fermer la connexion, une bonne pratique en C# est de la déclarer dans un bloc using, ce qui garantira la libération de l'objet à la sortie du bloc.

Vous pouvez également voir que le code a été protégé par un try-catch. La plupart du temps, l'exception sera une SqlException retournée par le serveur et e.Message contiendra le message d'erreur de SQL Server. En l'occurrence, la façon dont le catch renvoie ici le message puis la pile d'appels correspond à peu près au traitement par défaut de l'exception, en un peu moins bavard. Le but était simplement de vous rendre attentif au fonctionnement des exceptions SQL, qui peuvent vous être utiles en trappant les SqlException pour, par exemple, gérer vos transactions. Pour information, T-SQL comporte désormais un mécanisme de trappage d'erreur en try-catch.

La requête envoyée à SQL Server est simplement un SELECT dans une fonction (assimilée à une vue système) qui retourne une extraction des tables système, qui sont cachées à l'utilisateur en version 2005. Le but ici est de retourner la liste des sessions ouvertes par un utilisateur connecté.

Vous remarquerez la détection de la valeur nulle du paramètre à l'aide de la propriété IsNull de celui-ci, et l'utilisation de l'objet WindowsIdentity dans SqlContext pour retourner le nom du login connecté (l'équivalent de la fonction T-SQL system_user).

II-E. Quelques Notes

Comme avec une procédure stockée T-SQL, vous pouvez attribuer une valeur de retour en définissant votre méthode comme retournant un SqlInt32 (ou un System.Int32, ce qui entraînera une conversion implicite) à la place du void. N'essayez pas de définir un autre type de données, vous serez arrêté au déploiement, par l'échec du CREATE PROCEDURE. Il vous suffira d'indiquer votre valeur de retour par un return.

Pour définir un paramètre OUTPUT, passez votre paramètre par référence. En C#, ajouter le mot-clé ref devant la déclaration de votre paramètre. Vous devez également utiliser le mot-clé OUTPUT dans la déclaration de votre paramètre dans le CREATE PROCEDURE.

Comme vous ne pouvez pas définir de valeur par défaut aux paramètres des fonctions C#, les valeurs par défaut des paramètres se définissent au CREATE PROCEDURE. Ce mécanisme vous permet d'ailleurs de créer plusieurs procédures au nom différent appelant le même code .NET, avec des valeurs par défaut différentes.

Vous pouvez créer de toute pièce des resultsets dans votre code à l'aide des objets et méthodes suivants :

SqlMetaData	**Définir les colonnes de votre table et leur type de données**
SqlDataRecord	Créer des enregistrements, en passant un tableau de SqlMetaData au constructeur
Pipe.SendResultsStart(SqlDataRecord)	Commencer l'envoi du resultset au client
Pipe.SendResultsRow(SqlDataRecord)	Envoyer un SqlDataRecord
Pipe.SendResultsEnd()	Terminer l'envoi du resultset

II-F. Transactions

Pour gérer l'éventuelle transaction dans laquelle votre procédure stockée peut s'exécuter, vous avez à votre disposition l'objet Transaction du namespace System.Transactions (dans System.Transactions.dll), qui est une nouveauté du framework 2.0. L'intégration entre cet objet Transaction et les transactions SQL Server est très étroite. Lorsque du code .NET est exécuté dans le contexte du SQLCLR, l'objet System.Transactions.Transaction.Current reçoit le contexte de transaction. Si celui-ci est non nul, cela veut dire qu'une transaction est en cours. Tout code SQL exécuté dans la connexion de contexte, et même dans une autre connexion au serveur, est enlisté dans la transaction. En fait, cet objet est capable de convertir automatiquement la transaction locale en une transaction distribuée lorsqu'une connexion est ouverte vers un autre serveur via ADO.NET, ce qui la rend particulièrement puissante et permet de gérer les transactions distribuées avec un codage minimal.

III. Déploiement

Lorsque votre procédure est créée, il vous reste à l'intégrer dans SQL Server. Vous avez pour cela plusieurs choix:

- Si vous utilisez un autre outil de développement, comme par exemple SharpDevelop, vous pouvez ajouter l'assemblage généré, depuis SQL Server, soit visuellement à travers le Management Studio, soit avec la commande CREATE ASSEMBLY.
- Utiliser le déploiement intégré dans Visual Studio 2005.

Si vous utilisez un autre outil de développement, comme par exemple SharpDevelop, vous pouvez ajouter l'assemblage généré, depuis SQL Server, soit visuellement à travers le Management Studio, soit avec la commande CREATE ASSEMBLY.

Le déploiement intégré (clic droit sur le projet dans l'explorateur de solutions, *deploy*), va prendre toute la procédure en charge : copie sur le serveur, remplacement d'une éventuelle version existante, enregistrement de l'assemblage et de la procédure.

```
-- Enregistrement l'assemblage
CREATE ASSEMBLY GetSessions FROM 'c:\temp\Programmez.dll'
WITH PERMISSION_SET=SAFE
GO
-- Enregistrer la procédure
CREATE PROCEDURE GetSessions    @user varchar(255) AS EXTERNAL NAME [Programmez].[
StoredProcedures].[
GetSessions]
```

Vous pouvez rechargez la procédure avec un ALTER, seulement si vous n'avez pas modifié son interface publique, c'est-à-dire la signature de la méthode.

```
ALTER ASSEMBLY GetSessions FROM 'c:\temp\Programmez.dll'
```

Un conseil : vérifiez, par exemple à l'aide d'une trace, ce que fait l'outil de déploiement de Visual Studio. Vous pouvez préférer un déploiement manuel, par exemple pour ajuster le nom et la taille de vos paramètres, ou leur attribuer une valeur par défaut.

Après déploiement, votre procédure pourra être appelée depuis SQL Server à l'aide d'une commande semblable à :

```
exec dbo.GetSessions @user = 'mondomaine\moi' -- en cas de connexion intégrée Windows
```

Copyright © 2006 . Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée.

Responsable bénévole de la rubrique SQL-Server : David Barbarin (Mikedavem) - **Contacter par email**

Vos questions techniques : **forum d'entraide SQL-Server** - Publiez vos articles, tutoriels et cours
et rejoignez-nous dans l'équipe de rédaction du club d'entraide des développeurs francophones
Nous contacter - Hébergement - Participez - Copyright © 2000-2011 www.developpez.com - Legal informations.