



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# Les procédures stockées et les fonctions utilisateur



Grégory CASANOVA

# Sommaire

---

1	Introduction.....	3
2	Pré-requis .....	4
3	Les procédures stockées.....	5
3.1	Introduction aux procédures stockées.....	5
3.2	Gestion des procédures stockées.....	5
3.2.1	Création d'une procédure stockée.....	5
3.2.2	Modifier une procédure stockée.....	9
3.2.3	Suppression d'une procédure stockée .....	10
4	Les fonctions utilisateur .....	12
4.1	Introduction aux fonctions utilisateur.....	12
4.2	Création d'une fonction .....	13
4.3	Modification d'une fonction.....	16
4.4	Suppression d'une fonction.....	17
5	Conclusion .....	18

## 1 Introduction

Les bases de données sont utilisées partout. Comme simple conteneurs de données, comme entités capables de communiquer sur un réseau ou encore avec une application cliente distante. C'est cet aspect que nous allons valoriser dans ce chapitre : le fait que toute base de données puisse être exploitable depuis une application cliente. Dans cette démarche, les procédures stockées et les fonctions utilisateurs sont très utiles. Ces **objets** de la base de données, puisqu'ils sont référencés en tant que tels, ont la particularité d'être directement appelable depuis une application cliente. Nous allons détailler tous les avantages de ce type d'objets dans ce chapitre.

## 2 Pré-requis

Pour comprendre au mieux ce chapitre, vous devrez au préalable avoir vu :

- Les généralités sur les bases de données (Chapitre 2)
- La création des différents objets de la base vue auparavant (Chapitres 2, 3 et 5)
- Les généralités sur le code T-SQL et la programmabilité du langage (Chapitre 4)

## 3 Les procédures stockées

### 3.1 Introduction aux procédures stockées

Les procédures stockées sont des ensembles d'instructions du DML, pouvant être exécutés par simple appel de leur nom ou par l'instruction EXECUTE. Les procédures stockées sont de véritables programmes qui peuvent recevoir des paramètres, être exécutés à distance, renvoyer des valeurs et possédant leurs propres droits d'accès (EXECUTE). Celles-ci sont compilées une première fois, puis placées en cache mémoire, ce qui rend leur exécution plus performante du fait que le code soit précompilé. Les procédures stockées sont contenues dans la base de données, et sont appelable par leurs noms. Il existe une multitude de procédures stockées pré intégrées dans SQL Server lors de l'installation qui servent principalement à la maintenance des bases de données utilisateur. Celle-ci commence toujours par les trois caractères « sp\_ » comme stored procedure. Pour résumer les avantages des procédures stockées, nous allons lister leurs utilisations :

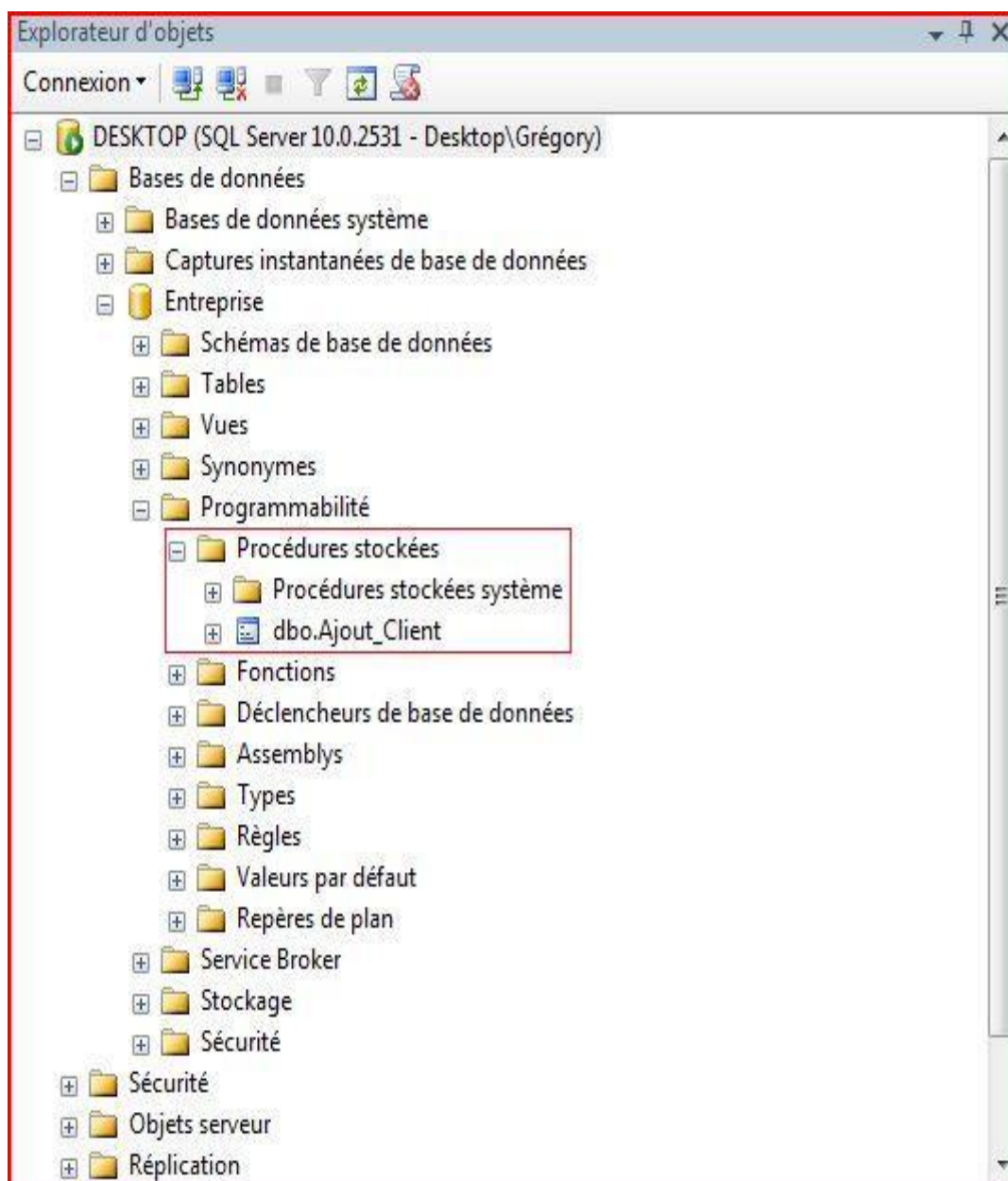
- **Accroissement des performances.**
- **Sécurité d'exécution.**
- **Possibilité de manipuler les données système.**
- **Implémente le traitement en cascade et l'enchaînement d'instructions.**

### 3.2 Gestion des procédures stockées

#### 3.2.1 Création d'une procédure stockée

Pour créer une procédure stockée, nous sommes obligés de passer par du code T-SQL, c'est pourquoi il est important de bien avoir lu le chapitre 4, traitant des généralités du code T-SQL. En revanche, il existe un assistant de génération automatique de la structure d'une procédure stockée. Nous allons tout d'abord étudier la structure générale d'une procédure stockée avec cette génération automatique, puis nous donnerons un exemple, présent dans le script de la base que nous utilisons, pour bien comprendre les notions exposées sur les procédures stockées.

Tout d'abord, pour générer le script automatiquement, étendez les nœuds de l'explorateur d'objet comme ceci :



On peut déjà remarquer la présence d'une procédure stockée, `Ajout_Client`, que vous devez aussi procéder si vous avez téléchargé le script de la base que nous utilisons pour notre cours, la base de données `Entreprise`. Maintenant, pour créer une nouvelle procédure stockée en générant le code automatiquement, il vous suffit de faire un click droit sur le nœud « Procédures stockées » et de choisir l'option « Nouvelle procédure stockée... ». Une nouvelle fenêtre de requête s'ouvre dans SSMS, vous proposant le code pour créer une nouvelle procédure stockée. Le code est le suivant :



```

-- =====
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date,,>
-- Description:    <Description,,>
-- =====
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> =
<Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> =
<Default_Value_For_Param2, , 0>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO

```

Nous allons maintenant détailler le code.

```
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
```

Nous créons une procédure stockée avec l'instruction DDL `CREATE PROCEDURE` suivie du nom à donner à la procédure. Ce nom vous permettra de l'appeler et de la reconnaître dans la base.

```

-- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> =
<Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> =
<Default_Value_For_Param2, , 0>

```

Nous devons ensuite préciser les variables que prend en paramètre la procédure stockée, durant son appel. Ces variables vont nous servir par la suite dans la définition des actions que la procédure stockée fait. Nous pouvons initialiser ou non les variables, le plus important est bien entendu de donner un nom conventionnel et un type de donnée à nos variables. Pour plus d'informations sur les variables, accédez au chapitre 4 sur le T-SQL, à la partie Variables.

```
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

Les instructions `AS BEGIN` et `END` sont les délimiteurs du code à utiliser par la procédure stockée. Toutes les instructions comprises entre ces deux mots clés seront prises en compte et exécutées par la procédure stockée.

Maintenant que nous avons présenté la structure générale de création d'une procédure stockée, prenons un exemple concret pour l'illustrer. Nous allons nous appuyer sur la procédure stockée déjà créée dans le script de notre base de données Entreprise, la procédure stockée `Ajout_Client`. Voici le code de création d'une procédure stockée d'ajout client :

```
CREATE PROCEDURE Ajout_Client
@Nom varchar(50), @Prenom varchar(50),
@Numero varchar(50), @Adresse varchar(50),
@Mail varchar(50)
AS
BEGIN
    INSERT INTO Client
    (Nom_Client,
    Prenom_Client,
    Numero_Client,
    Adresse_Client,
    Mail_Client)
    VALUES
    (@Nom,
    @Prenom,
    @Numero,
    @Adresse,
    @Mail)
END
GO
```

On remarque parfaitement que l'on définit les paramètres avec des variables en début de code, après l'instruction `CREATE PROCEDURE`. Ensuite, entre les blocs `AS BEGIN` et `END`, nous faisons un simple dans la table `INSERT INTO`, `Client` et nous passons les paramètres de notre procédure stockée en valeur, après l'instruction `VALUES`. Il est alors possible d'utiliser cette procédure stockée de la façon suivante :

```
EXEC dbo.Ajout_Client 'NARBONNE', 'Christophe',
33678764534, '17 allée des embrumes', NULL
```

L'instruction `EXEC` ou `EXECUTE` permet d'exécuter la procédure stockée. Il est alors nécessaire d'entrer des valeurs pour les paramètres indiqués dans la procédure stockée.

**Remarque :** Il est obligatoire de mettre les arguments de la procédure stockée dans l'ordre dont elles sont décrites dans la procédure stockée. Si toute fois vous ne voulez pas les mettre dans l'ordre, il est possible de préciser le nom de la variable et d'y assigner une valeur de la façon suivante :

```
EXEC dbo.Ajout_Client @Nom = 'NARBONNE', @Prenom = 'Christophe',
@Numero = 33678764534, @Mail = NULL, @Adresse = '17 allée des embrumes'
```

Dans les deux cas, le résultat est le même puisque une ligne est ajoutée à la table Client. Voici le résultat :

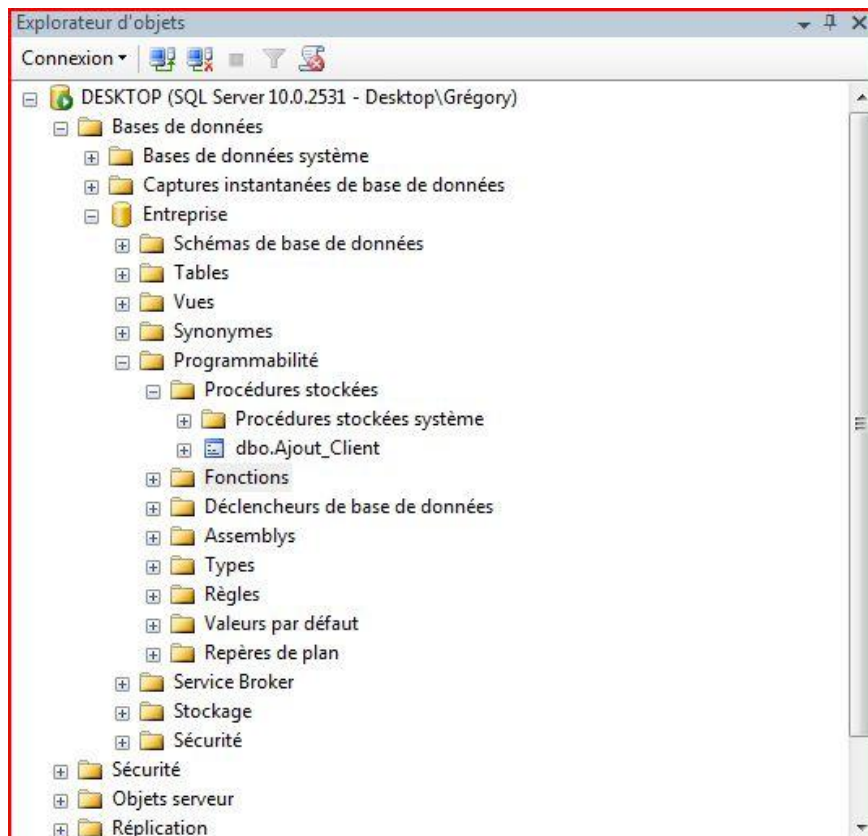
	Id_Client	Nom_Client	Prenom_Client	Numero_Client	Adresse_Client	Mail_Client
1	1	CASANOVA	Grégory	33563456764	31 place de la chance	75554@supinfo.com
2	2	RAVILLE	James	33567876435	34 Avenue de le paix	James.Ravaille@Domaine.fr
3	3	DOLLON	Julien	33563765514	17 Rue du cotton	Jul.Dollon@dotnetfrance.com
4	4	VERGNAULT	Bertrand	33567654323	26 rue de la vie	Bertrand@Sondomaine.com
5	5	VASSELON	Jean Christophe	33567654342	19 avenue du sac	JCVD@leursdomaine.com
6	6	HOLLEBEC	Mathieu	33578763450	26 place des canons	MathHol@votredomaine.com
7	7	NARBONNE	Christophe	33678764534	17 allée des embrumes	NULL

Après la déclaration de création de procédure, et la déclaration des paramètres, il est possible de définir des options grâce à une clause WITH ou une clause FOR. Ces options sont les suivantes :

- **WITH RECOMPILE** : La procédure sera recompilée à chaque exécution.
- **WITH ENCRYPTION** : Permet de crypter le code dans la table système.
- **FOR REPLICATION** : Permet de préciser que la procédure sera utilisée lors de la réplication.

### 3.2.2 Modifier une procédure stockée

La modification d'une procédure stockée ne peut se faire que par l'interface graphique en premier lieu. Pour en modifier une, étendez tous les nœuds qui mènent à une procédure stockée en particulier, comme ceci :



Pour modifier une procédure stockée en particulier, il vous suffit alors de procéder à un click droit sur celle-ci, et de choisir l'option « Modifier ». Une nouvelle fenêtre de requête apparait, avec le code que vous aviez entré lors de la première création de votre procédure. Modifiez-le alors comme voulu et ré exécutez le pour modifier la procédure.

### 3.2.3 Suppression d'une procédure stockée

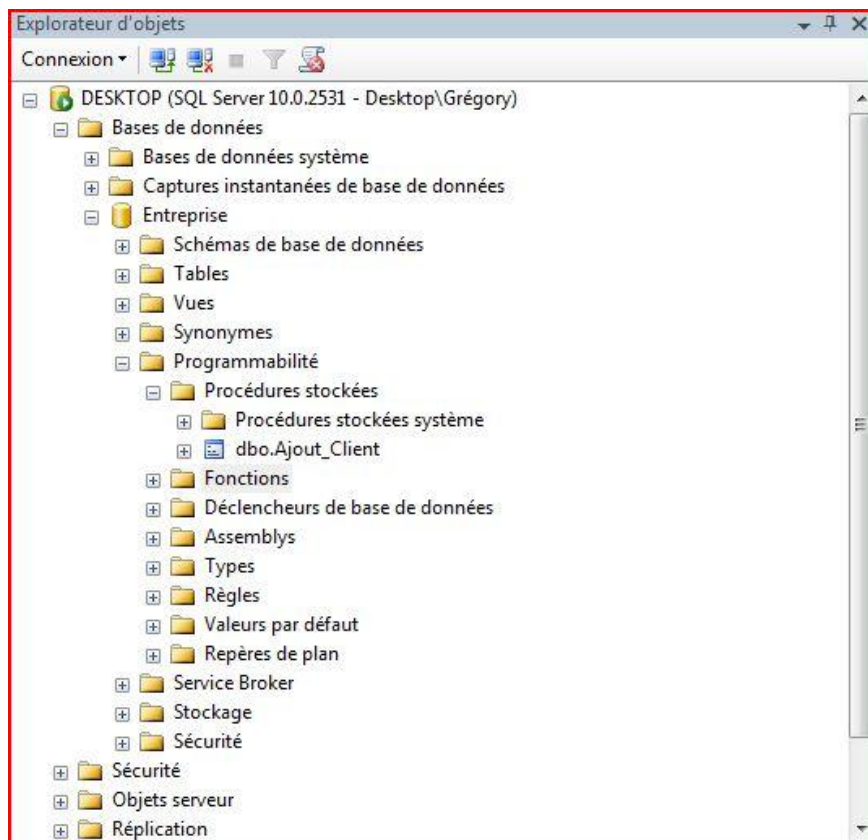
La suppression d'une procédure peut se faire en revanche de deux manières, avec du code ou l'interface graphique. Nous allons présenter les deux.

- Avec du code, il est très simple de supprimer une procédure stockée. Regardons la syntaxe générale :

```
USE Entreprise
DROP PROCEDURE Ajout_Client
```

Dans un premier temps, avec l'instruction `USE`, placez vous dans la base contenant la procédure stockée à supprimer. Si vous ne le faites pas, la procédure stockée sera indiqués comme un élément non existant dans la base par SQL Server. Utilisez ensuite les mots clé `DROP PROCEDURE` suivit du nom de la procédure stockée pour supprimer celle-ci. Exécutez le code. Vous venez de supprimer votre procédure stockée.

- Avec l'interface graphique, nous allons procéder comme pour une modification de la procédure. Etendez dans un premier temps tous les nœuds qui mènent à la procédure stockée à supprimer, comme ceci :



Opérez alors un simple click droit sur la procédure stockée choisie, et sélectionnez l'option « Supprimer ». Une nouvelle fenêtre apparait. Il s'agit simplement d'une validation de votre choix, aucune autre option n'est disponible. Cliquez sur « OK ». Votre procédure stockée est supprimée.

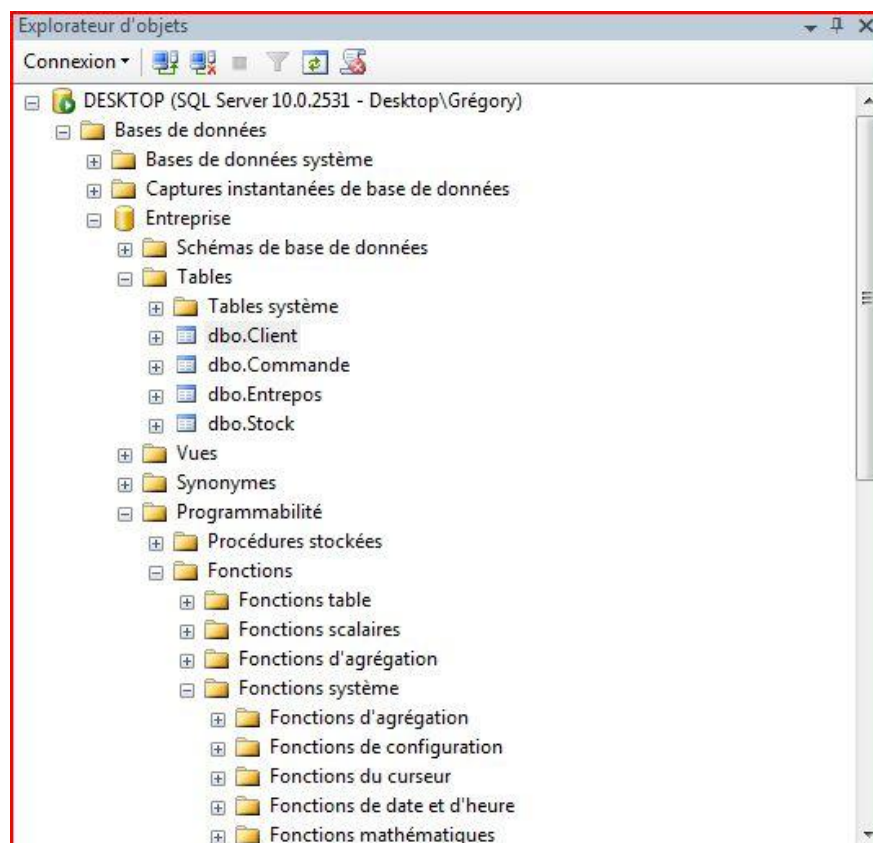
## 4 Les fonctions utilisateur

### 4.1 Introduction aux fonctions utilisateur

Les fonctions utilisateurs sont de plusieurs types. Trois pour être précis. Il y a les fonctions scalaires, les fonctions tables en ligne et les fonctions multi-instructions. Une fonction peut accepter des arguments, et ne peut retourner que deux types de données : une valeur scalaire ou une table.

- Les fonctions scalaires retournent, grâce au mot clé RETURN, une valeur scalaire. Tous les types de données peuvent être retournés par une fonction scalaire hors mis timestamp, table, cursor, text, ntext et image.
- Les fonctions table ne retournent comme résultat, qu'une table, qui est le résultat d'une instruction SELECT.

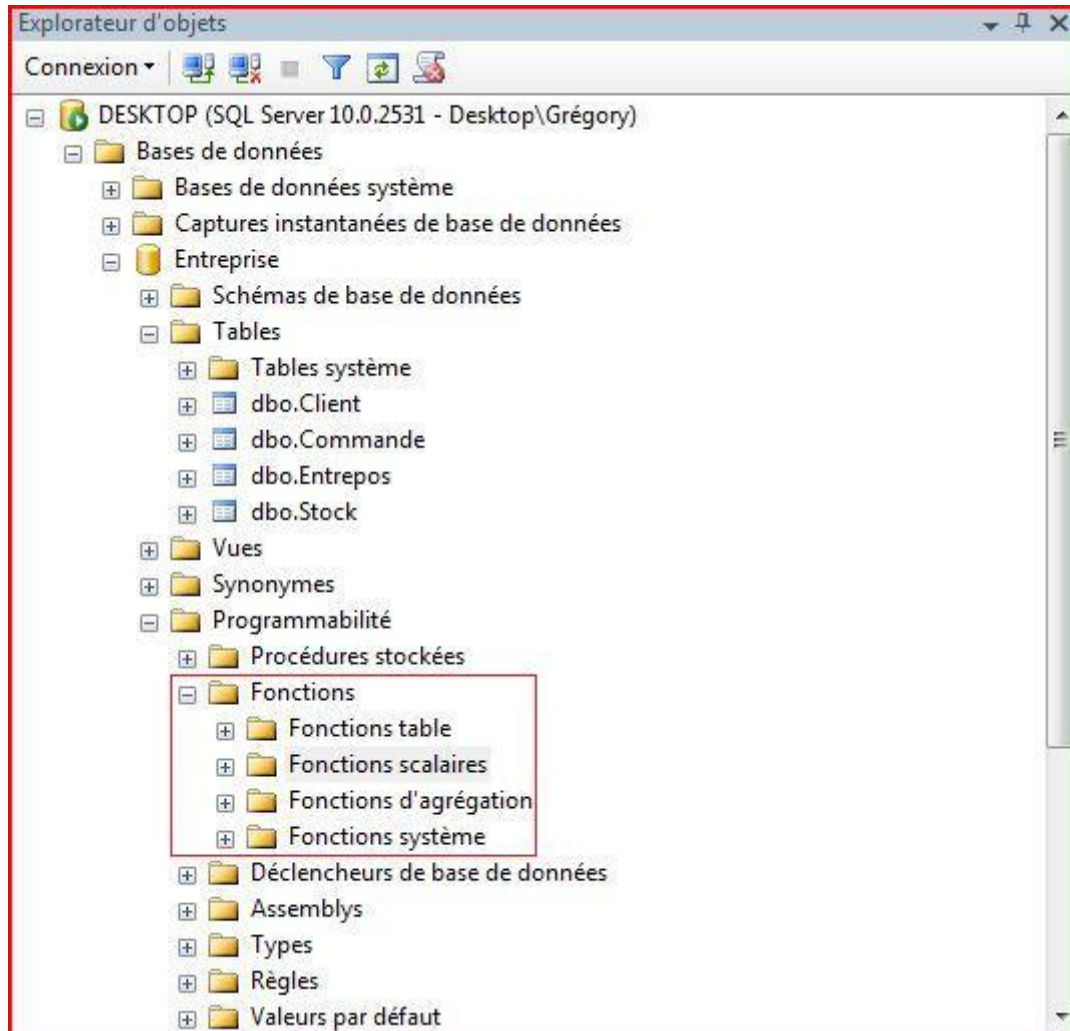
Le champ d'action d'une fonction est vraiment limité puisqu'il n'est possible de modifier que des objets locaux à la fonction. Il n'est pas possible de modifier des objets de la base, contenus à l'extérieur de la fonction. La création d'une fonction se fait par l'instruction DDL CREATE FUNCTION, habituelle lors de la création d'objets dans la base. De plus, il existe une multitude de fonctions système, utilisables par simple appel de celle-ci. Elles sont disponibles dans l'explorateur d'objets ainsi.



Détaillons maintenant tous les types de fonctions existants.

## 4.2 Création d'une fonction

La création d'une fonction se fait quasiment de la même manière pour les trois types. Nous allons bien sûr présenter les trois avec trois exemples, simplement nous allons le faire dans la même partie. Voici les deux syntaxes principales de création de fonction (scalaire et table). Nous pouvons y accéder en dépliant les nœuds de l'explorateur d'objet jusqu'au nœud « Fonctions », appliquer un click droit sur le type de fonction à créer et choisir l'option « Nouvelle fonction... ».



- **Fonction table :**



```

-- =====
-- Template generated from Template Explorer using:
-- Create Inline Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date,,>
-- Description:    <Description,,>
-- =====
CREATE FUNCTION <Inline_Function_Name, sysname, FunctionName>
(
    -- Add the parameters for the function here
    <@param1, sysname, @p1> <Data_Type_For_Param1, , int>,
    <@param2, sysname, @p2> <Data_Type_For_Param2, , char>
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT 0
)
GO

```

Les fonctions table et table multi-instructions sont différentes dans le sens où le format de retour est différent. En effet, la fonction table retourne la solution d'une requête SELECT, alors que la fonction table multi-instruction, retournera une variable de type table, contenant l'instruction SELECT opérée par la fonction.

Prenons deux exemples pour mieux comprendre la syntaxe :

- **Fonction table simple :**

```

CREATE FUNCTION recommander_stock (@Id int, @seuil int)
RETURNS TABLE
AS
RETURN (SELECT * FROM Stock WHERE Id_Stock = @Id AND Quantite < @Seuil)

```

Dans ce cas là, après l'instruction de création de fonction, `CREATE FUNCTION`, on indique simplement que la fonction retourne une donnée de type table avec la clause `RETURNS TABLE`. Par

la suite, après la clause `AS`, on précise quelle instruction doit être retournée dans la valeur de retour de type table de la fonction.

- **Fonction table multi-instruction :**

```
CREATE FUNCTION table_multi (@Id int)
RETURNS @variable TABLE (Id_Stock int, Quantite int, Nom_Entrepos
varchar(25))
AS
BEGIN
SELECT @variable = (SELECT Id_Stock, Quantite, Nom_Entrepos
FROM Entrepos E INNER JOIN Stock S
ON E.Id_Entrepos = S.Id_Entrepos
WHERE S.Id_Stock = @Id)
RETURN
END
```

Dans ce cas, en revanche, la valeur de retour est toujours une table, simplement, on donne à une variable ce type et on définit les colonnes qu'elle contient. Les valeurs définies par le `SELECT` y seront alors contenues. Il faut faire attention, en contrepartie, à ce que le nombre de colonnes dans la variable retournée par la fonction ait le même nombre de colonnes que le résultat retourné par l'instruction `SELECT`.

- **Fonction scalaire :**

```

-- =====
-- Template generated from Template Explorer using:
-- Create Scalar Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date, ,>
-- Description:    <Description, ,>
-- =====
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
    -- Add the parameters for the function here
    <@Param1, sysname, @p1> <Data_Type_For_Param1, , int>
)
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
    -- Declare the return variable here
    DECLARE <@ResultVar, sysname, @Result> <Function_Data_Type, ,int>

    -- Add the T-SQL statements to compute the return value here
    SELECT <@ResultVar, sysname, @Result> = <@Param1, sysname, @p1>

    -- Return the result of the function
    RETURN <@ResultVar, sysname, @Result>

END
GO

```

Ici, on crée une fonction scalaire que l'on reconnaît grâce à l'instruction `RETURNS int`. On se sert bien entendu de l'instruction `CREATE FUNCTION` suivie du nom de la fonction à utiliser et des paramètres à prendre en compte entre parenthèses. Les clauses `AS BEGIN` et `END` déclarent dans l'ordre, le début et la fin de la fonction. Cette fonction retourne une valeur qui désigne le nombre d'article présents, pour un stock passé en paramètre.

Dans les fonctions, certaines options sont disponibles, comme dans les procédures stockées. Elles vont vous permettre certaines actions sur cette fonction, et celle-ci sont appelables directement après la définition du type de retour de la fonction :

- **WITH SCHEMABINDING** : Cette option permet de lier la fonction à tous les objets de la base auxquels elle fait référence. Dans ce cas, la suppression et la mise à jour de n'importe quel objet de la base, lié à la fonction est impossible.
- **WITH ENCRYPTION** : Permet de crypter le code dans la table système.

### 4.3 Modification d'une fonction

La modification d'une fonction n'est possible que par une instruction T-SQL DDL, l'instruction `ALTER FUNCTION`. En effet, dans ce cas là, les commandes `CREATE FUNCTION` et `ALTER FUNTION`

auront quasiment la même fonction, puisque dans les deux cas, tout le corps de la fonction doit être réécrit et totalité. Il n'est pas possible d'ajouter ou supprimer seulement une seule ligne dans celui-ci. Leur seule différence réside donc dans le fait que la fonction soit créée ou modifiée. Voyons donc la syntaxe qui permet la modification d'une fonction en Transact SQL :

```
ALTER FUNCTION nombre_element_stock (@Entrepos int)
RETURNS int
AS
BEGIN
DECLARE @nb int
SELECT @nb = COUNT(Id_Stock)
FROM Stock
WHERE Id_Entrepos = @Entrepos
RETURN @nb
END
GO
```

On remarque donc aisément que la seule modification qu'il existe entre la création et la modification d'une fonction, réside dans le remplacement du mot clé `CREATE FUNCTION` par le mot clé `ALTER FUNCTION`.

#### 4.4 Suppression d'une fonction

La suppression d'une fonction peut, comme pour les procédures stockées ou tout autre objet de la base, être faite de deux manières. La première est la méthode graphique. Nous ne détaillerons pas cette méthode, puisqu'elle est la même pour tout objet de la base. Nous rappellerons juste que il vous suffit d'étendre, dans votre explorateur d'objet, les nœuds correspondants au chemin de votre fonction, de faire un click droit sur celle-ci, et de sélectionner « Supprimer ».

Avec le langage Transact SQL, la méthode est aussi la même que pour tout autre objet de la base. Nous utiliserons l'instruction `DROP` et nous l'adapterons au cas d'une fonction. Voici la méthode type de suppression d'une fonction par code T-SQL :

```
DROP FUNCTION nombre_element_stock
```

L'instruction `DROPFUNCTION` nous permet donc de supprimer une fonction, de n'importe quel type. Il suffit juste de préciser le nom de la fonction après cette instruction.

## 5 Conclusion

Durant ce chapitre, nous avons présenté deux nouveaux objets de la base : les procédures stockées et les fonctions. Retenez bien que les procédures stockées ne sont que des segments de code qui portent un nom, et qui sont précompilés, ce qui améliore et leur fiabilité, et leur performance. En revanche les fonctions sont des entités capables d'agir dans leur environnement direct, et donc pas capable de modifier une table. Dans le prochain chapitre, nous verrons les triggers qui se décomposent en deux parties : Les triggers ou déclencheurs du DML et ceux du DDL. Ce sont eux qui constituent le début de la programmation dite événementielle dans SQL Server 2008.