T-SQL

Table des matières

[1 Introduction 2](#_Toc480363860)

[2 Procédures stockées et fonctions 2](#_Toc480363861)

[3 Gestion des erreurs 4](#_Toc480363862)

[3.1 Description d’une erreur 4](#_Toc480363863)

[3.2 Interception d’une erreur 5](#_Toc480363864)

[3.3 Levée d’une erreur 5](#_Toc480363865)

[4 Métadonnées 7](#_Toc480363866)

[5 SQL dynamique 10](#_Toc480363867)

# Introduction

Transact-SQL (abrégé en T-SQL) est une extension du langage SQL pour le SGBD SQL Server

Il comprend des instructions de type langage procédural, telles que :

* Déclaration et affectation de variables
* Lots et blocs d’instructions
* Instructions d’affichage
* Instructions conditionnelles
* Boucles
* Fonctions
* Procédures stockées
* Déclencheurs
* …

Le T-SQL est surtout utilisé à l’intérieur de procédures et fonctions stockées directement dans la base de données.

Les avantages de la centralisation de code SQL dans la base, plutôt que dans l’application sont les suivants :

* Cela peut faciliter la réutilisabilité du code SQL
* Cela facilite la mise à jour du code SQL, sans avoir à recompiler ni déployer l’application entière
* Cela permet d’avoir une couche d’isolation supplémentaire entre le code de l’application et la structure des données dans la base. Ainsi, si l’application accède à certaines données uniquement depuis des vues, procédures ou fonctions, l’appel de ces dernières peut rester inchangé, même si leur code SQL interne change, par exemple suite à une modification du schéma de la base.
* Cela offre des possibilités supplémentaires pour la sécurisation des données, car on peut appliquer des droits sur les différents objets de la base.
* Dans le cas d’un accès à la base par l’intermédiaire du réseau, cela permet de diminuer le trafic réseau puisque les instructions SQL à transmettre au SGBD sont beaucoup plus courtes.

# Procédures stockées et fonctions

Les procédures stockées et fonctions sont des ensembles d’instructions T-SQL remplissant un objectif précis, et stockés dans la base.

La première fois qu’elles sont exécutées, SQL Server crée un plan d’exécution et le met en cache pour optimiser les exécutions suivantes.

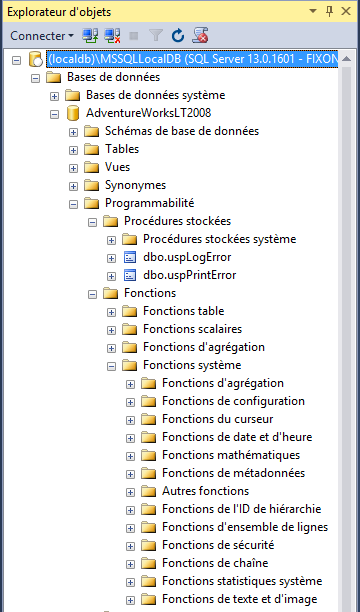
Les procédures et fonctions peuvent retourner une valeur ou une table. Une fonction retourne obligatoirement un résultat, alors que ce n’est pas le cas pour une procédure.

Une fonction s'utilise essentiellement dans une requête SQL alors qu'une procédure est un programme autonome qui peut être lancé par un client.

A ce titre une fonction ne peut pas comporter :

* De transaction (puisque la requête qui l'utilise est déjà une transaction)
* De mise à jour (INSERT, UPDATE, DELETE...)
* De gestion d'erreur à la manière d'une procédure
* De SQL dynamique
* D'appel à une procédure
* De création de tables temporaires
* D'ordre SQL DDL ou DCL, exception faite des fonctions table pour lesquels les ordres CREATE, INSERT etc., visent uniquement la variable table

SQL Server fournit plusieurs centaines de procédures stockées et fonctions prédéfinies. Dans Management Studio, elles sont visibles dans les branches « Programmabilité \ Procédures stockées \ Procédures stockées système » et « Programmabilité \ Fonctions \ Fonctions système », comme le montre l’image ci-dessous :



# Gestion des erreurs

## Description d’une erreur

Lorsqu’une erreur survient lors de l’exécution d’un code T-SQL, SQL Server renvoie un message d’erreur contenant les informations suivantes :

* Numéro identifiant
* Niveau (= sévérité) : numéro compris entre 0 et 24, de l’erreur la moins grave à la plus grave
* Etat : numéro qui identifie le contexte dans lequel s’est produite l’erreur
* Nom de la procédure ou fonction qui a provoqué l’erreur
* Numéro de la ligne qui a provoqué l’erreur (par rapport au début du lot courant)
* Texte descriptif

Exemple 1 :

Essayons de créer la procédure suivante :

create procedure usp\_UpdateCategory

as

begin

update Categories set CategoryID = 99 where CategoryID = 1

end

Lors de l’exécution de ce code, on obtient l’erreur suivante :

Msg 8102, Niveau 16, État 1, Procédure usp\_UpdateCategory, Ligne 4 [Ligne de départ du lot 10]

Cannot update identity column 'CategoryID'.

En effet, le code ci-dessus essaie d’affecter explicitement une valeur à la colonne CategoryId de type auto-incrémenté, ce qui n’est pas autorisé. Ici l’erreur a été détectée dès la création de la procédure

Exemple 2 :

Créons la fonction suivante :

create function ufn\_DivideNumbers (@a float, @b float)

returns float

as

begin

return @a / @b

end

go

L’exécution de ce code ne provoque aucune erreur.

Exécutons maintenant le code suivant :

select dbo.ufn\_DivideNumbers (5, 0)

Il provoque l’erreur suivante :

Msg 8134, Niveau 16, État 1, Ligne 17

Divide by zero error encountered.

## Interception d’une erreur

Comme en C#, il est possible d’intercepter une erreur afin d’y réagir de façon appropriée. Ceci est possible au moyen d’une construction BEGIN TRY…END TRY, BEGIN CATCH…END CATCH

Exemple :

begin try

select dbo.ufn\_DivideNumbers (5, 0)

end try

begin catch

print 'Erreur lors de l''appel à ufn\_DivideNumbers : ' + ERROR\_MESSAGE()

end catch

Ce code affiche le message suivant :

Erreur lors de l'appel à ufn\_DivideNumbers : Divide by zero error encountered.

A l’intérieur d’un bloc catch, on peut utiliser les fonctions suivantes pour récupérer les informations de l’erreur :

ERROR\_NUMBER(), ERROR\_SEVERITY(), ERROR\_STATE(), ERROR\_LINE(), ERROR\_MESSAGE()

On peut également choisir de propager l’erreur au niveau supérieur si on n’est pas en mesure de la gérer correctement, grâce à l’instruction THROW

On peut ainsi choisir de traiter certaines erreurs et pas d’autres ;

Exemple :

begin try

select dbo.ufn\_DivideNumbers (5, 0)

end try

begin catch

if ERROR\_NUMBER() = 8134

print 'Le diviseur doit être différent de 0'

else

THROW

end catch

Ici, on n’intercepte que l’erreur de division par 0 et on propage toutes les autres.

Comme en C#, les erreurs sont propagées tant qu’elles ne sont pas interceptées.

## Levée d’une erreur

On peut lever soi-même une erreur en utilisant l’instruction RAISERROR.

Exemple :

create procedure usp\_DeleteOrderDetail @orderId int, @productId int

as

begin

-- Si on tente de supprimer la dernière ligne de la commande,

-- on lève une erreur

if (select count(\*) from Order\_Details where OrderID = @orderId) = 1  
 begin  
 RAISERROR ('Vous ne pouvez pas supprimer la dernière ligne de la commande !',

12, 1)  
 return  
 end

-- Suppression de la ligne de commande pour le N° de commande

-- et le N° de produit passés en paramètre

delete from Order\_Details

where OrderID = @orderId and ProductID = @productId

end

go

/!\ Attention, RAISERROR ne fait pas sortir de la procédure ; il faut appeler explicitement return pour sortir.

Pour que le code appelant puisse intercepter spécifiquement l’erreur qu’on a levée, il est préférable de lui associer un numéro identifiant. Pour cela, il faut au préalable définir un message d’erreur à l’aide de la procédure sp\_addmessage :

exec sp\_addmessage @msgnum = 50001, @severity = 12,

@msgText = 'Last order detail can not be delete', @lang='us\_english'

exec sp\_addmessage @msgnum = 50001, @severity = 12,

@msgText = 'La dernière ligne de la commande ne peut pas être supprimée',  
 @lang='french'

Caractéristiques des messages d’erreurs

* Le message doit obligatoirement être déjà défini en anglais avant d’être défini dans une autre langue.
* Le numéro doit être compris entre 50001 et 2^31. Pour remplacer un message déjà existant, il faut spécifier en plus le paramètre @replace = ‘replace’

Utilisation du message pour lever l’erreur :

create procedure usp\_DeleteOrderDetail @orderId int, @productId int

as

begin

-- Si on tente de supprimer la dernière ligne de la commande,

-- on lève une erreur

if (select count(\*) from Order\_Details where OrderID = @orderId) = 1  
 begin

RAISERROR (50001, 12, 1)

RETURN  
 end

-- Suppression de la ligne de commande pour le N° de commande

-- et le N° de produit passés en paramètre

delete from Order\_Details

where OrderID = @orderId and ProductID = @productId

end

Pour tester, exécutons le code suivant :

exec usp\_DeleteOrderDetail 10271, 33

On obtient bien le message d’erreur :

Msg 50001, Niveau 12, État 1, Procédure usp\_DeleteOrderDetail, Ligne 7 [Ligne de départ du lot 36]

Last order detail can not be delete

NB/ Pour modifier la langue d’affichage des messages, d’erreur, on peut au préalable faire :

set language 'french'

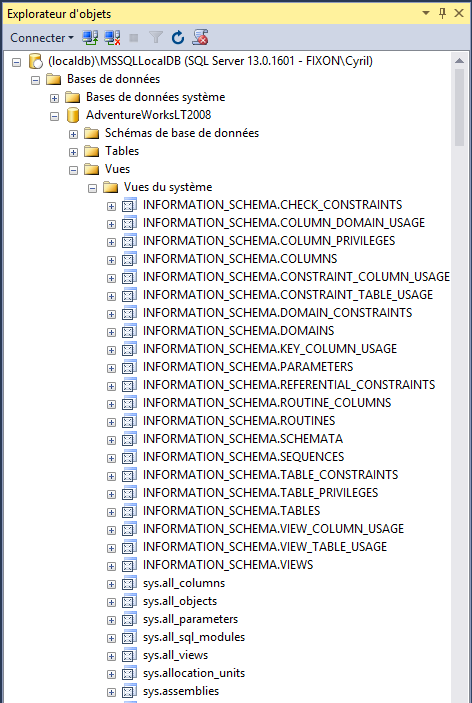
# Métadonnées

Une base de données est autodescriptive, c’est-à-dire qu’elle contient des métadonnées qui décrivent les objets qu’elle contient.

La façon d’accéder aux métadonnées d’une base fait l’objet d’une norme ISO. Celle-ci définit un schéma nommé INFORMATION\_SCHEMA, contenant un ensemble de vues, qui permet un accès standardisé aux métadonnées, quel que soit la version de la base, et la façon dont les métadonnées sont organisées et stockées.

NB/ On peut faire l’analogie avec les métadonnées des objets d’une application net, qui sont décrites par la classe System.Type, et accessibles durant l’exécution, grâce à la méthode GetType de la classe Object.

SQL Server est conforme à la norme ISO qui définit INFORMATION\_SCHEMA. Dans Management Studio, les vues sont visibles dans la branche « Vues \ Vues du système » de chaque base :



En outres, comme on peut le voir dans l’image ci-dessus, SQL Server a ses propres vues système définies dans le schéma sys.

Il est conseillé d’utiliser au maximum les vues standard de INFORMATION\_SCHEMA, qui sont indépendantes de la version de la base. Ces vues sont décrites sur [cette page MSDN](https://msdn.microsoft.com/fr-fr/library/ms186778.aspx).

Les vues du schéma sys fournissent cependant plus d’informations. Pour les exploiter, on utilise souvent les fonctions **OBJECT\_ID** et **OBJECT\_NAME**, qui permettent d’obtenir l’id d’un objet à partir de son nom et inversement En effet, les vues du schéma sys ne fournissent souvent que les id et non les libellés des objets.

NB/ Une instance SQL Server contient également des bases de données système qui contiennent d’autres types de métadonnées :

* La base Master contient les métadonnées de l’instance (paramétrages, utilisateurs, référence des bases de données)
* La base msdb contient les métadonnées des services annexes (SQL-Agent, messagerie, …)

Voici quelques exemples de requêtes exploitant les vues standard et les vues système de SQL Server :

--Liste des colonnes d'une base triées par schéma et par table :

select TABLE\_SCHEMA, TABLE\_NAME, COLUMN\_NAME, DATA\_TYPE, CHARACTER\_MAXIMUM\_LENGTH, IS\_NULLABLE, COLUMN\_DEFAULT

from INFORMATION\_SCHEMA.COLUMNS

order by 1,2,3

--Liste des clés étrangères avec tables et colonnes associées :

SELECT DISTINCT OBJECT\_NAME(f.parent\_object\_id) [Table]

,COL\_NAME(fc.parent\_object\_id, fc.parent\_column\_id) [Colonne clé étrangère]

,OBJECT\_NAME (f.referenced\_object\_id) [Table référencée]

,COL\_NAME(f.parent\_object\_id, fc.referenced\_column\_id) [Colonne Id]

FROM sys.foreign\_keys f

INNER JOIN sys.foreign\_key\_columns AS fc ON f.object\_id = fc.constraint\_object\_id

-- Nombre d’enregistrements par table :

SELECT SCHEMA\_NAME(t.schema\_id) [SchemaName],

t.name AS [TableName],

SUM(p.Rows) AS [RowCount]

FROM sys.tables AS t

INNER JOIN sys.partitions AS p ON t.object\_id = p.object\_id

WHERE p.index\_id < 2

GROUP BY t.schema\_id , t.name

ORDER BY [SchemaName], [TableName]

Explications :

Le nombre de lignes de la table est égal au nombre de lignes de l'index cluster associé. Comme une table peut s’étendre sur plusieurs partitions, il faut faire la somme des nombres de lignes de l’index sur toutes les partitions.

Index\_id représente l'ID de l'index et peut prendre les valeurs suivantes :

* 0 : segment de mémoire
* 1 : index cluster
* >= 2 : index non-cluster

# SQL dynamique

Il est possible de construire et exécuter une requête dynamiquement, c’est-à-dire par le code lui-même. Pour cela, SQL Server fournit la procédure stockée sp\_executesql qui facilite beaucoup les choses.

Exemple :

DECLARE @req AS NVARCHAR(500)

DECLARE @TypeParam AS NVARCHAR(500)

-- Construction de la requête avec paramètres

SET @req = 'SELECT \* FROM Employees WHERE EmployeeID = @EmpID'

-- Spécification du paramètre avec son type

SET @TypeParam = '@EmpID SMALLINT'

-- Exéécution de la requête en fournissant une valeur de paramètre

EXECUTE sp\_executesql @req, @TypeParam, 1

-- Requête avec plusieurs paramètres

SET @req = 'SELECT \* FROM Employees WHERE City = @Ville and year(BirthDate) > @AnneeNais'

Set @TypeParam = '@Ville nvarchar(15), @AnneeNais SMALLINT'

Execute sp\_Executesql @req, @TypeParam,'London', 1960

NB/ SQL Server fournit également la procédure stockée sp\_MSforeachtable, qui permet d’exécuter du sql dynamique sur chacune des tables de la base.

Exemple : pour supprimer toutes les tables de la base :

EXEC sp\_MSforeachtable "drop table ?"