

Les procédures et les curseurs avec MS-SQL Server

par [Baptiste Wicht \(home\)](#)

Date de publication : Le 23 Octobre 2006

Dernière mise à jour : Le 18 Novembre 2006

Cet article vous permettra d'approfondir vos connaissances sur les procédures (et fonctions) stockées et sur les curseurs.

- I - Introduction
- II - Procédures stockées
- III - Fonctions
- IV - Curseurs
 - IV-A - Fonctions de curseurs
 - IV-B - Ensembliste VS Curseur
- V - Conclusion
 - V-A - Remerciements
 - V-B - Liens

I - Introduction

Vous avez parfois entendu parler de fonctions, de procédures ou de curseurs, mais jamais vous n'avez vraiment su de quoi il en retourne ? Alors cet article est fait pour vous. Nous allons passer en revue toutes ces différentes choses et vous apprendre comment les utiliser et à quoi elles servent.

II - Procédures stockées

Une procédure stockée est un bout de code que l'on met sur le serveur et qui va s'exécuter ensuite directement sur le serveur. Ce code est déjà compilé, donc pour des appels nombreux de la même méthode, on ne doit pas tout recompiler à chaque fois.

Une procédure est un bout de code qui ne aucune valeur, il se contente d'effectuer un travail.

Syntaxe de création de procédure :

```
CREATE PROCEDURE name @parameter1 type1, @parameter2 type2 #  
AS sql_statement
```

Explication de la syntaxe

- name : C'est tout simplement le nom que l'on veut donner à notre procédure
- @parameter : c'est un paramètre que l'on passe à la procédure quand on l'appelle
- sql_statement : c'est un jeu d'instruction SQL que la procédure va effectuer quand elle sera appelée

Exemple : On a besoin d'une procédure qui additionne deux nombres passé en paramètre, qui les additionne et qui imprime le résultat :

```
CREATE PROCEDURE print_addition @nbre1 INT, @nbre2 INT  
AS PRINT @nbre1 + @nbre2
```

Appel : Pour appeler une procédure, il suffit tout simplement d'écrire EXECUTE suivi du nom de votre procédure :

```
EXECUTE print_addition 1, 2
```

On peut noter qu'il n'est pas obligatoire d'utiliser EXECUTE, on peut aussi utiliser EXEC ou alors directement le nom de la fonction (mais seulement si c'est la première du lot).

III - Fonctions

Une fonction est une procédure qui renvoie une valeur.

Syntaxe de création :

```
CREATE FUNCTION name ([@parameter1 type1, @parameter2 type #])
RETURNS type
BEGIN
    sql_statement
RETURN @value
END
```

Explications de la syntaxe

- name : le nom de la fonction
- @parameterX : un paramètre de la fonction
- typeX : le type du paramètre
- type : le paramètre de retour de la fonction
- sql_statement : les instructions SQL que l'on va effectuer
- @value : la valeur que va retourner la fonction

Exemple : On veut faire une fonction qui prenne deux nombres en paramètres, qui les additionne et qui renvoie le résultat :

```
CREATE FUNCTION return_addition (@nbre1 INT, @nbre2 INT)
RETURNS INT
BEGIN
    RETURN @nbre1 + @nbre2
END
```

Appel : Pour appeler une fonction, il faut utiliser son nom à deux composantes (owner.name), il suffit de l'utiliser tel quel :

```
PRINT dbo.return_addition(1,3)
```

Cela va nous afficher 4.

IV - Curseurs

Le curseur est un mécanisme de mise en mémoire en tampon permettant de parcourir les lignes d'enregistrements du résultat renvoyé par une requête. Les curseurs sont envoyés par MS-SQL Server tout le temps, mais on ne voit pas le mécanisme se passer, ainsi lors d'une requête SELECT, SQL Server va employer des curseurs.

Pour utiliser un curseur, il faut commencer par le déclarer :

```
DECLARE name CURSOR FOR  
SELECT #
```

On peut aussi l'utiliser avec de la modification en ajoutant FOR UPDATE à la fin de la requête, bien que ce ne soit pas conseillé.

Ensuite, il faut ouvrir ce curseur avec OPEN name et ne pas oublier de le fermer à la fin avec CLOSE name. Il faut aussi utiliser DEALLOCATE pour libérer la mémoire du curseur.

Pour récupérer les valeurs actuelles contenues dans le curseur, il faut employer :

```
FETCH name INTO @value1, @value2 #
```

Cela va stocker les valeurs actuelles de l'enregistrement courant dans les variables @valueX, qu'il ne faut surtout pas oublier de déclarer.

On peut néanmoins utiliser FETCH pour d'autres choses :

- Aller à la première ligne : FETCH FIRST FROM curseur_nom
- Aller à la dernière ligne : FETCH LAST FROM curseur_nom
- Aller à la ligne suivante : FETCH NEXT FROM curseur_nom
- Aller à la ligne précédente : FETCH PRIOR FROM curseur_nom
- Aller à la ligne X : FETCH ABSOLUTE ligne FROM curseur_nom
- Aller à X lignes plus loin que l'actuelle : FETCH RELATIVE ligne FROM curseur_nom

Pour parcourir un curseur, on peut employer une boucle WHILE qui teste la valeur de la fonction @@FETCH_STATUS qui renvoie 0 tant que l'on n'est pas à la fin.

```
DECLARE @nom VARCHAR(50)  
  
DECLARE curseur_auteurs CURSOR FOR  
SELECT auteur_nom FROM t_auteurs  
  
OPEN curseur_auteurs  
  
FETCH curseur_auteurs INTO @nom  
  
WHILE @@FETCH_STATUS = 0  
BEGIN  
    PRINT @nom  
    FETCH curseur_auteurs INTO @nom  
END
```

```
CLOSE curseur_auteurs  
DEALLOCATE curseur_auteurs
```

On peut bien entendu imbriquer plusieurs curseurs les uns dans les autres pour des choses plus compliquées.

Concrètement, maintenant que nous avons vu comment fonctionnait un curseur et comment l'employer, que fait-il de plus qu'une simple requête ? Il permet surtout d'intervenir sur le résultat de la requête. On peut intervenir sur chaque valeur retournée, on peut modifier ces valeurs ou supprimer des lignes. On peut aussi réaliser des opérations avec ces données avant qu'elles arrivent au programme qui les utilise, c'est à dire des calculs de somme, des maximums, des modifications de date, des formatages de chaînes de caractères.

Un exemple intéressant est le parcours avec rupture, c'est à dire parcourir et si on a déjà eu une fois cet objet on ne le réaffiche pas. Dans l'exemple que je vais vous présenter, on affiche tous les genres, les auteurs par genre et pour chaque auteur les livres qu'ils ont écrits. On emploie des ruptures pour vérifier que l'on n'a pas déjà affiché une fois cet élément :

```
DECLARE @titre VARCHAR(50), @genre VARCHAR(50), @rupture_genre VARCHAR(50), @rupture_auteur  
        VARCHAR(50), @auteur VARCHAR(50)  
DECLARE @i_genre INT  
  
SET @i_genre = 1  
SET @rupture_genre = ''  
SET @rupture_auteur = ''  
  
DECLARE curseur_ouvrages CURSOR FOR  
SELECT ouvrage_titre, genre_nom, auteur_nom FROM t_ouvrages O  
LEFT OUTER JOIN t_genres  
    ON genre_id = ouvrage_genre  
LEFT OUTER JOIN t_ouvrages_auteurs TOA  
    ON TOA.ouvrage_id = O.ouvrage_id  
LEFT OUTER JOIN t_auteurs TA  
    ON TA.auteur_id = TOA.auteur_id  
ORDER BY ouvrage_genre  
  
OPEN curseur_ouvrages  
  
FETCH curseur_ouvrages INTO @titre , @genre, @auteur  
  
WHILE @@FETCH_STATUS = 0  
BEGIN  
    IF @genre != @rupture_genre  
    BEGIN  
        PRINT ''  
        PRINT CONVERT(CHAR(2),@i_genre) + '. ' + @genre  
  
        SET @i_genre = @i_genre + 1  
        SET @rupture_auteur = ''  
    END  
  
    IF @auteur != @rupture_auteur  
    BEGIN  
        PRINT ''  
        PRINT @auteur  
        PRINT '-----'  
    END  
  
    PRINT @titre;  
  
    SET @rupture_genre = @genre  
    SET @rupture_auteur = @auteur  
    FETCH curseur_ouvrages INTO @titre , @genre, @auteur
```

```
END
```

```
CLOSE curseur_ouvrages  
DEALLOCATE curseur_ouvrages
```

IV-A - Fonctions de curseurs

Il y a trois fonctions intéressantes concernant les curseurs :

- **@@FETCH_STATUS** : Renvoie l'état de la dernière instruction FETCH effectuée sur un curseur. Elle renvoie 0 si tout s'est bien passé, -1 s'il n'y a plus de lignes et -2 si la ligne est manquante.
- **@@CURSOR_ROWS** : Renvoie le nombre de lignes se trouvant actuellement dans le dernier curseur ouvert. Renvoie 0 s'il n'y a pas de curseurs ouverts ou plus de ligne dans le dernier curseur. Renvoie un nombre négatif si le curseur a été ouvert de manière asynchrone (voir config de SQL Server)
- **CURSOR_STATUS** : Nous permet de vérifier qu'une procédure a bien renvoyé un curseur avec un jeu de données. Je ne vais pas m'étendre sur cette fonction, vu sa complexité, référez-vous à la doc si vous en avez besoin.

IV-B - Ensembliste VS Curseur

La manipulation ensembliste est une juste une requête qui va nous renvoyer un ensemble de données (un resultset). C'est tout simplement des requêtes SELECT. Ces requêtes sont simples à effectuer bien qu'on puisse aller assez loin avec elles. Malheureusement on ne dispose pas vraiment de pouvoir sur elles, c'est la base de données qui décide ce qu'elle va nous renvoyer.

La lecture par curseur est en fait la face cachée de la manipulation ensembliste, dès que l'on fait un SELECT, la base de données va employer des curseurs pour construire le résultat à notre requête. Comme on vient de le voir, on peut employer ces curseurs nous-mêmes pour avoir plus de souplesses. Par contre, les curseurs sont réputés comme étant assez instables et en les manipulant nous-mêmes, on s'expose à des risques plus élevés qu'un simple SELECT.

Ensembliste

- Très simple à utiliser
- Aucune possibilité de modification sur le retour
- Risques quasi-nuls
- Très recommandés

Curseurs

- Pas recommandé, à n'utiliser que dans des cas où l'on ne peut rien faire d'autres
- Assez complexe à utiliser
- Très puissant
- Risques d'instabilité
- Pouvoir complet sur le retour puisque c'est nous qui faisons tout

V - Conclusion

Les procédures stockées et les fonctions peuvent se révéler très pratique si c'est une portion de code que vous exécutez souvent. Les curseurs peuvent aussi être très pratique de par leur souplesse, néanmoins, il faut les utiliser à bon escient et le moins souvent possible, et surtout ne pas oublier de les fermer après toute utilisation.

V-A - Remerciements

Je tiens à remercier mon maître de stage qui m'a tout d'abord appris tout ça et ensuite relu cet article.

V-B - Liens

Télécharger la documentation officielle de SQL Server

