

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer des composants d'accès
aux données »

SQL Server et Transact SQL
Le langage DML partie 3 - Déclencheurs

Apprentissage

Mise en pratique

Evaluation

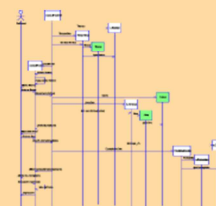
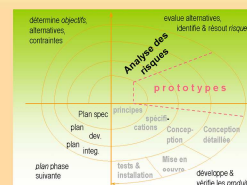
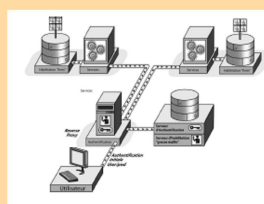


TABLE DES MATIERES

Table des matières	3
1. UTILISATION DES DECLENCHEURS DML.....	6
1.1 Définitions	6
1.2 Usage des déclencheurs.....	6
1.3 Règles lors de la conception de déclencheurs :	7
2. PRINCIPE DE FONCTIONNEMENT.....	8
2.1 Illustration des mécanismes concernant les 'tables' deleted et inserted :	9
2.2 Syntaxe :	11
2.3 Dans SQL Server Management Studio	11
2.4 Les options :	11
3. Des exemples :	12
4. MODIFICATION / SUPPRESSION	14
5. VISUALISATION D'INFORMATIONS.....	14
6. IMBRICATION DES DECLENCHEURS.....	15
7. Et pour aller plus loin : manipulation des curseurs	16
7.1 Déclaration d'un curseur	16
7.1.1 Portée.....	16
7.1.2 Défilement	16
7.1.3 Type du curseur.....	17
STATIC	17
KEYSET	17
DYNAMIC.....	17
7.1.4 Verrouillages des lignes.....	18
READ_ONLY.....	18
SCROLL_LOCKS	18
OPTIMISTIC.....	18
7.1.5 Génération du jeu de résultats.....	18
7.1.6 Opérations de mise à jour.....	18
UPDATE [OF Colonne 1 [...Colonne n]].....	18
7.2 Ouverture et lecture	19
7.2.1 Lecture	19

7.3	Un exemple à la loupe	20
7.3.1	Déclaration	20
7.3.2	Processus de parcours du curseur	20
7.3.3	Fermeture et libération mémoire	21
7.4	Processus des curseurs en bref	21

Objectifs

Après étude de ce document, vous serez à même de comprendre le fonctionnement des déclencheurs et leur programmation dans SQL server.

Pré requis

Connaître les bases du langage de requête SQL.

Maîtriser les bases de la programmation procédurale.

Avoir pris connaissance des instructions de programmation en langage Transact-SQL.

Outils de développement

SQL Server Management Studio.

Méthodologie

Vous découvrirez tout d'abord le fonctionnement interne de SQL Server lors de la mise à jour des données. Vous apprendrez ensuite à utiliser les 'tables' spéciales `inserted` et `deleted` pour valider ou non ces mises à jour et vous apprendrez comment annuler une mise à jour invalide.

Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

Ressources

L'aide en ligne de SQL Server disponible à tout moment et indexée par rapport au contexte (touche F1).

Lectures conseillées

1. UTILISATION DES DECLENCHEURS DML

1.1 DEFINITIONS

Un déclencheur DML (Data Manipulation Language) est un **type spécial de procédure stockée** qui présente trois caractéristiques :

- Il est **associé à une table**
- Il **s'exécute automatiquement** lorsque un utilisateur essaie de modifier des données par une instruction DML (update, delete, insert) sur la table ou il est défini.
- Il **ne peut être appelé directement**.

Un déclencheur est aussi appelé couramment **Trigger**.

Le déclencheur et l'instruction qui a provoqué son exécution sont traités comme une seule transaction. Les définitions de déclencheurs peuvent inclure une instruction `ROLLBACK TRANSACTION` même en l'absence d'instruction `BEGIN TRANS` explicite car une opération de mise à jour est implicitement une transaction.

Les déclencheurs servent à **maintenir une intégrité des données de bas niveau** et non à envoyer des résultats de requête. Leur avantage principal réside dans le fait qu'ils peuvent contenir une logique de traitement complexe. **Ils doivent être employés lorsque les contraintes déclaratives n'offrent pas les fonctionnalités nécessaires.**

1.2 USAGE DES DECLENCHEURS

On pourra utiliser les déclencheurs, par exemple pour :

- **Modifier en cascade les données** des tables reliées dans une base de données
- **Mettre en œuvre une intégrité des données plus complexe qu'une contrainte CHECK**
A la différence des contraintes `CHECK`, les déclencheurs peuvent référencer des colonnes d'autres tables. Par exemple, dans une gestion commerciale, lorsque la commande d'un article est passée, une ligne est insérée dans la table `Lignes_de_Commandes`. Un déclencheur `INSERT` sur cette table pourra déterminer si la commande peut être livrée ou non, en examinant la colonne quantité en stock dans la table `Stock`. Si cette valeur est insuffisante, il pourra générer automatiquement un ordre de commande fournisseur et avertir le gestionnaire.
- **Renvoyer des messages d'erreur personnalisés**
les règles, les contraintes et les valeurs par défaut ne peuvent communiquer des erreurs que par l'intermédiaire des messages d'erreur système standards.

1.3 REGLES LORS DE LA CONCEPTION DE DECLENCHEURS :

- Les déclencheurs sont réactifs alors que les contraintes ont un caractère préventif. Les contraintes sont contrôlées en premier, les déclencheurs sont exécutés en réponse à une instruction `INSERT`, `UPDATE` ou `DELETE`.
- Les tables peuvent avoir plusieurs déclencheurs pour une même action. Par exemple, plusieurs déclencheurs `INSERT` peuvent être définis pour une même table, mais ils doivent être indépendants les uns des autres.
- Il n'est pas possible de créer des déclencheurs sur des vues, mais les déclencheurs peuvent référencer des vues.
- Les déclencheurs ne doivent pas renvoyer de jeux de résultats.
- Le propriétaire de la table et les membres des rôles `db_owner`, `db_dlladmin` et `sysadmin` peuvent créer et supprimer des déclencheurs sur une table. De plus le créateur du déclencheur doit avoir la permission d'exécuter toutes les instructions sur toutes les tables. Si l'une des permissions est refusée, la transaction est annulée totalement.
- **Deux 'tables' spéciales sont disponibles pendant l'exécution des déclencheurs** : la 'table' **DELETED** contient les copies des lignes affectées par les instructions `INSERT`, `DELETE` et `UPDATE`, la 'table' **INSERTED** contient les copies des lignes affectées par l'instruction `INSERT` et `UPDATE` qui vient de se produire sur la table (en effet, une mise à jour est considérée comme une suppression de l'image *avant* et une insertion de l'image *après*). Ces 'tables' peuvent être référencées par une instruction `SELECT` ou utilisées pour tester des valeurs, mais ne peuvent être modifiées. Ces 'tables' spéciales, virtuelles, **ne sont disponibles que le temps de l'exécution du trigger** et présentent au développeur, sous forme de tables, un extrait bien utile du journal des transactions concernant uniquement la table concernée.

2. PRINCIPE DE FONCTIONNEMENT

Les étapes suivantes montrent comment un déclencheur est lancé lors d'un événement sur la table où il est défini.

Cas 1 : événement **INSERT**

- Une instruction **INSERT** est exécutée sur une table comportant un déclencheur **INSERT**
- L'instruction **INSERT** est journalisée dans le journal des transactions
la table **inserted** 'reçoit' la copie des lignes ajoutées à la table (la table et la table **inserted** ont des lignes en commun)
- Le déclencheur est lancé et ses instructions s'exécutent

Cas 2 : événement **DELETE**

- Une instruction **DELETE** est exécutée sur une table comportant un déclencheur **DELETE**
- L'instruction **DELETE** est journalisée dans le journal des transactions
la table **deleted** 'reçoit' la copie des lignes supprimées de la table (la table et la table **deleted** n'ont aucune ligne en commun)
- Le déclencheur est lancé et ses instructions s'exécutent

Cas 3 : événement **UPDATE**

- Une instruction **UPDATE** est exécutée sur une table comportant un déclencheur **UPDATE**
- L'instruction **UPDATE** est journalisée dans le journal des transactions sous la forme **INSERT** et **DELETE**
la table **deleted** 'reçoit' la copie des lignes de la table représentant l'image avant la modification.
la table **inserted** 'reçoit' la copie des lignes de la table représentant l'image après la modification
- Le déclencheur est lancé et ses instructions s'exécutent

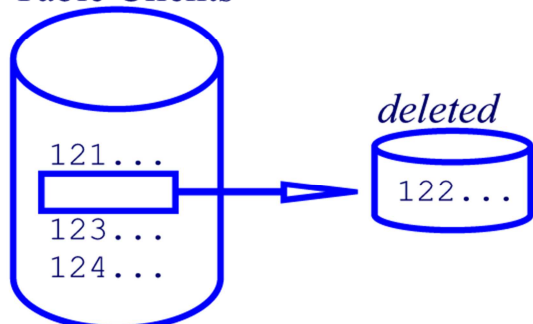
2.1 ILLUSTRATION DES MECANISMES CONCERNANT LES 'TABLES' DELETED ET INSERTED :

(ceci concerne les triggers par défaut, c'est-à-dire de type AFTER – voir plus loin)

✓ Cas de suppression de lignes de table (instruction SQL delete)

Suppression client 122

Table Clients

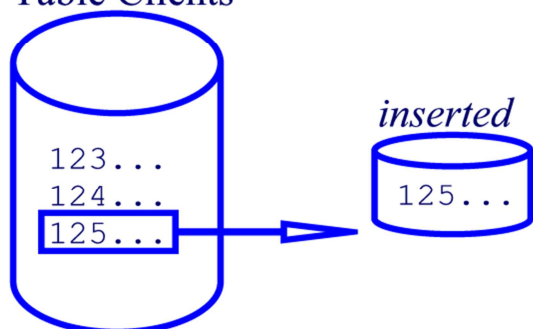


La/les lignes supprimées sont placées dans la table temporaire DELETED et supprimées de la table réelle ; la table DELETED et la table mise à jour *ne peuvent pas avoir de lignes en commun*.

✓ Cas de création d'une ligne de table (instruction SQL insert)

Insertion client 125

Table Clients

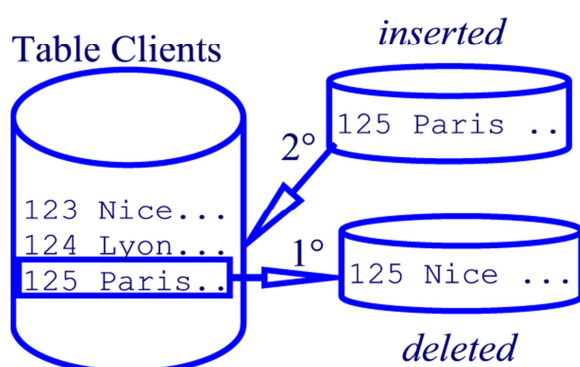


La/les lignes nouvelles sont placées dans la table temporaire INSERTED **et** dans la table réelle ; *toutes les lignes de la table INSERTED apparaissent aussi dans la table mise à jour*.

Modification client 125 (quitte Nice pour Paris)

✓ Cas de modification d'une ligne de table (instruction SQL update)

Table Clients

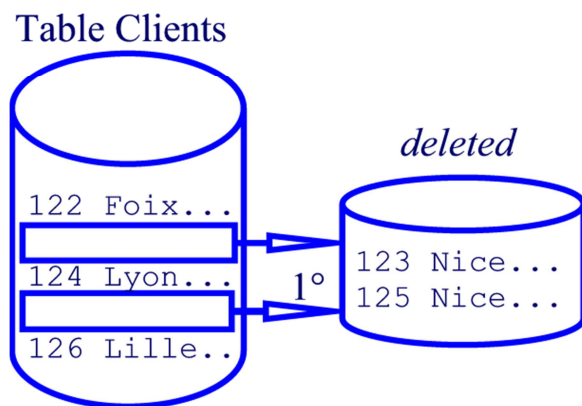


La/les lignes avant modification sont placées dans la table temporaire DELETED et la/les lignes après modification sont placées dans la table temporaire INSERTED et dans la table réelle. Un update revient en fait à un delete accompagné d'un insert.

Attention : les tables virtuelles peuvent contenir plusieurs lignes, en fonction de la mise à jour effectuée

✓ Exemple de suppressions multiples suivant une condition logique :

Suppression clients niçois



Dans cet exemple, le trigger en suppression sur une table des clients permet de rechercher toutes les commandes associées aux clients supprimés et de les supprimer de la table des commandes

```
create trigger cli_supp on client for delete as
begin
    delete from commande where noclient in (select noclient
        from deleted)
end
```

opérateur **IN** et non **=** car il peut y avoir eu plusieurs clients supprimés par la même commande delete

CREATION DE DECLENCHEURS

A l'intérieur du code d'un déclencheur, SQL Server proscrit l'utilisation des instructions :

- CREATE, ALTER, DROP (création, modification ou suppression d'objet)
- GRANT, REVOKE et DENY (gestion des droits)
- LOAD et RESTORE
- RECONFIGURE
- TRUNCATE TABLE
- UPDATE STATISTICS
- SELECT INTO (car elle crée une table)

La création de déclencheurs s'effectue à l'aide de l'instruction **CREATE TRIGGER**. Cette instruction spécifie la table sur laquelle le déclencheur est défini, l'événement provoquant son exécution et les instructions qu'il contient.

<https://docs.microsoft.com/fr-fr/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15>

2.2 SYNTAXE :

```
CREATE TRIGGER nom_déclencheur
ON table | view
[WITH ENCRYPTION | EXECUTE as Clause]
{ FOR | AFTER | INSTEAD OF }{[INSERT][,][UPDATE][,][DELETE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS instructions_SQL /
[ ENCRYPTION ]
[ EXECUTE AS Clause ]
```

2.3 DANS SQL SERVER MANAGEMENT STUDIO

Les déclencheurs DML dont l'étendue est une table de base de données figurent dans le dossier **Déclencheurs**, situé dans chaque table de la base de données correspondante.

2.4 LES OPTIONS :

AFTER joue le même rôle que **FOR**, indiquant que le déclencheur va s'exécuter lorsque toutes les opérations spécifiées dans l'instruction SQL de déclenchement ont été réalisées avec succès (fonctionnement par défaut de SQL Server dit « optimiste » c'est-à-dire qu'il considère a priori qu'il n'y aura pas d'erreur).

Avec les déclencheurs **AFTER**, l'instruction appelante est faite, et peut être annulée.

Les déclencheurs **AFTER** ne peuvent être définis sur des vues.

Les instructions SQL d'un déclencheur **AFTER** ou **FOR INSERT** vont s'exécuter à chaque **insertion** sur la table, les instructions SQL d'un déclencheur **AFTER** ou **FOR UPDATE** à chaque

Programmation SQL serveur - Déclencheurs

modification d'une ligne de la table, les instructions SQL d'un déclencheur **AFTER** ou **FOR DELETE** à chaque **suppression** d'une ligne de la table.

Un même déclencheur peut être créé pour 2 événements différents si les traitements à effectuer sont similaires (ou assez proches) : `CREATE TRIGGER nom_de_trigger ON nom_de_table FOR INSERT, UPDATE AS ...`

L'option **ENCRYPTION** empêchera les utilisateurs d'afficher le texte des déclencheurs.

La clause **EXECUTE AS** spécifie le contexte de sécurité dans lequel la fonction définie par l'utilisateur est exécutée (voir Sécurité dans SQL Server).

3. DES EXEMPLES :

Les triggers sont souvent utilisés pour effectuer des contrôles sur les données mises à jour dans les tables. En cas de détection d'erreur, il est bien entendu nécessaire d'abandonner la mise à jour en cours. SQL Serveur a un principe de fonctionnement 'optimiste', c'est-à-dire qu'il écrit les modifications sur disque avant de déclencher l'exécution du trigger (sauf pour les triggers de type `instead of`) ; les schémas précédents montrent ce mécanisme. Le langage Transact-SQL nous offre donc l'instruction **ROLLBACK TRANSACTION** pour 'défaire ce qui vient d'être fait' (voir exemple 2 ci-dessous).

Exemple 1: Création d'un déclencheur **AFTER Vente_Insert** sur l'instruction **INSERT** de la table **Ventes** de structure

`VENTES (vnt_art, vnt_cli, vnt_qte, vnt_prix)`

Lorsqu'une ligne est insérée dans la table **Ventes**, le déclencheur décrémente la colonne quantité en stock dans la table **ARTICLES** de la quantité vendue ;

`ARTICLES (art_num, art_nom, art_coul, art_pa, art_pv, art_qte, art_frs)`

```
CREATE TRIGGER Vente_Insert
ON Ventes
FOR INSERT
AS
BEGIN
    UPDATE Article SET Art_Qte = Art_Qte - Vnt_Qte
    FROM Article INNER JOIN Inserted
    ON Art_Num. = Vnt_Art
END
```

Dans l'exemple ci-dessus, la table **inserted** contient la ligne de **Ventes** qui vient d'être ajoutée, et les colonnes de la table **Vente** sont manipulables à travers la table **inserted**.

Un déclencheur peut être défini avec l'instruction **IF UPDATE(...)**, qui contrôle que la mise à jour concerne bien **une colonne donnée**.

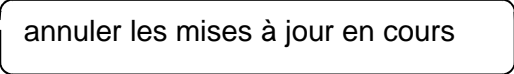
Si une action **FOR UPDATE** est spécifiée, la clause **IF UPDATE(colonne)** peut être utilisée pour orienter l'action vers une colonne spécifique qui est mise à jour.

Exemple 2: Contrôle que le code département de la table `Employés`, de structure :

`EMPLOYES (NOEMP, NOM, PRENOM, DEPT, SALAIRE)`

n'a pas été modifié lors de l'opération d'update :

```
CREATE TRIGGER Employés_Dept_Upd
ON Employés
FOR UPDATE
AS
BEGIN
    IF UPDATE(DEPT)
    BEGIN
        RAISERROR ('Le n° de département ne peut pas être modifié', 10, 1)
        ROLLBACK TRAN
    END
END
```



Il est même possible de tester plusieurs colonnes :

`IF UPDATE (colonne1) AND|OR UPDATE (colonne2)`

On obtiendra des résultats similaires avec la clause `COLUMNS_UPDATED()`

Le principe de fonctionnement des déclencheurs **INSTEAD OF** est simple : l'instruction appelante est **interceptée, donc non réellement exécutée**, et le code du déclencheur la **remplace** : il est toujours possible de tester les valeurs insérées, mises à jour ou supprimées pour décider de la suite des opérations.

Les déclencheurs **INSTEAD OF** peuvent être définis sur des vues : un déclencheur sur une vue permet d'étendre la possibilité de mise à jour de vues multi tables ; un seul déclencheur **INSTEAD OF** par instruction est autorisé sur une table ou vue.

Exemple 3 : Création d'un déclencheur **INSTEAD OF Insert_Multiple** sur l'instruction **INSERT** de la vue multi tables, `Vue_TousClients` qui regroupent les clients français et étrangers.

Lorsqu'une ligne est insérée, le déclencheur met à jour les tables concernées `ClientsF` et `ClientsE`.

```
CREATE TRIGGER Insert_Multiple
ON Vue_TousClients
INSTEAD OF INSERT
AS
BEGIN
```

```

If (select payC from inserted)='F'
    Insert ClientsF select * from inserted
Else
    Insert ClientsE select * from inserted
END

```

inserted et deleted sont mises à jour, mais pas la table/vue

4. MODIFICATION / SUPPRESSION

Un déclencheur peut être modifié sans avoir à être supprimé, grâce à l'instruction **ALTER TRIGGER** : la définition modifiée remplacera la définition existante.

```

ALTER TRIGGER nom_déclencheur
ON table / View
[WITH ENCRYPTION]
FOR {[INSERT][,][UPDATE][,][DELETE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS
bloc_instructions_SQL

```

Un déclencheur peut être activé ou désactivé : un déclencheur désactivé n'est pas lancé lors de l'instruction INSERT, UPDATE ou DELETE.

```

ALTER TABLE table
{ENABLE | DISABLE} TRIGGER
{ALL | nom_déclencheur[,...n]}

```

Un déclencheur peut être supprimé par l'instruction **DROP TRIGGER** ; la suppression d'une table entraîne la suppression de tous les déclencheurs associés.

5. VISUALISATION D'INFORMATIONS

Pour obtenir des informations supplémentaires sur tous les déclencheurs, vous pouvez utiliser les procédures stockées système suivantes ou exécuter SQL Server Management Studio.

Procédure stockée	Informations
Sp_helptext nom_décl	Affiche le texte du déclencheur si non cryptée
Sp_depends nom_décl	Enumère les objets référencés par le déclencheur
Sp_helptrigger nom_table	Renvoie la liste des déclencheurs définis sur la table spécifiée

6. IMBRICATION DES DECLENCHEURS

Les déclencheurs peuvent être imbriqués, c'est à dire qu'un déclencheur modifiant une table pourra activer un autre déclencheur, qui pourra à son tour activer un autre déclencheur...etc. Les déclencheurs peuvent avoir jusqu'à 32 niveaux d'imbrication. Il est possible de connaître le niveau d'imbrication en cours grâce à la variable système @@NESTLEVEL.

L'**imbrication** est une fonctionnalité puissante qui peut servir à maintenir l'intégrité des données. Cette fonction activée à l'installation de SQL Server peut être désactivée occasionnellement (dans ce cas, si un déclencheur modifie une table, aucun déclencheur de cette table ne sera activé).

L'imbrication peut être activée / désactivée à l'aide de la procédure stockée système **sp_configure** ou à partir de la page **Paramètres du serveur** dans la boîte de dialogue **Propriétés** de SQL Server de SQL Server Management Studio.

Déclencheurs récursifs : Un déclencheur peut contenir une instruction UPDATE, INSERT ou DELETE affectant la même table ou une autre table.

Lorsque l'option base de données **recursive triggers** est activée, un déclencheur qui modifie des données dans une table provoque à nouveau son lancement dans une exécution récursive.

L'option **recursive triggers** est désactivée par défaut lors de la création de la base de données, mais elle peut être activée au moyen de la procédure stockée système **sp_dboption**, ou à partir de la page **Options**, dans la boîte de dialogue **Propriétés de la base de données** de SQL Server Management Studio.

Ces fonctionnalités restent à manier avec précaution...

7. ET POUR ALLER PLUS LOIN : MANIPULATION DES CURSEURS

Ce chapitre traite de la manipulation de données au travers des curseurs. Les curseurs permettent de parcourir un jeu de données issu d'une requête de sélection et de traiter individuellement chaque ligne.

En règle générale, nous traitons les lignes par lot. Toutes les lignes du lot font l'objet d'un traitement similaire.

Mais nous avons besoin, parfois, de différencier les opérations à exécuter en fonction de la valeur des colonnes d'une ligne.

Nous allons découvrir dans ce chapitre les différentes instructions relatives aux curseurs :

- La déclaration du curseur
- L'ouverture et la fermeture
- Le parcours du curseur
- La destruction du curseur

Ce support n'a pas pour objectif d'approfondir les techniques de manipulation de curseurs mais d'apporter la connaissance strictement nécessaire.

7.1 DECLARATION D'UN CURSEUR

Un curseur est une variable d'un type particulier CURSOR. Il se déclare donc à l'aide du mot clé **DECLARE**.

7.1.1 Portée

Il est possible de définir la portée du curseur.

Par défaut, la portée est locale à la procédure stockée ou au lot d'instructions dans lequel celui-ci est utilisé.

Mais la durée de vie d'un curseur peut être étendue à la durée de la connexion SQL qui l'a créée. Dans ce cas, il sera considéré comme de portée globale. Les mots-clés LOCAL, GLOBAL permettent de définir la portée du curseur.

Dans le cas d'un curseur de portée globale, la mémoire occupée par ce curseur sera libérée implicitement lors de la fermeture de la connexion SQL.

7.1.2 Défilement

Mots clés FORWARD_ONLY et SCROLL

Par défaut un curseur est en défilement vers l'avant uniquement. C'est le cas le plus fréquent d'utilisation : on lit le curseur séquentiellement depuis la première ligne jusqu'à la dernière au moyen de l'opération FETCH NEXT.

Mais il est possible d'étendre les fonctionnalités de parcours de ce dernier en complétant la définition du curseur à l'aide du mot clé **SCROLL**.

Il sera alors possible d'utiliser de multiples opérations de déplacement, de manière relative ou absolue, par le biais des instructions **FIRST**, **LAST**, **PRIOR**, **NEXT**, **RELATIVE** et **ABSOLUTE**.

7.1.3 Type du curseur

Les différents types de curseurs impactent les performances et le reflet des modifications apportées aux lignes présentes dans le curseur. Par défaut, le curseur est dynamique et reflète donc toutes les modifications apportées en cours de traitement sur les lignes présentes dans le curseur.

STATIC

Définit un curseur qui fait une copie temporaire des données qu'il doit utiliser. Toutes les réponses aux requêtes destinées au curseur sont effectuées à partir de cette table temporaire dans `tempdb`; par conséquent, les modifications apportées aux tables de base ne sont pas reflétées dans les données renvoyées par les extractions de ce curseur, et ce dernier n'accepte pas de modifications.

KEYSET

Spécifie que l'appartenance au curseur et l'ordre des lignes sont fixés lors de l'ouverture du curseur.

L'ensemble des clés qui identifient de manière unique les lignes est créé dans une table de `tempdb`, connue sous le nom de `keyset`.

Les modifications apportées aux valeurs non-clés dans les tables de la base, que ce soit celles effectuées par le propriétaire du curseur ou celles validées par les autres utilisateurs, sont visibles lorsque le propriétaire parcourt le curseur.

Les insertions effectuées par d'autres utilisateurs ne sont pas visibles. Si vous supprimez une ligne, une tentative d'extraction de la ligne renvoie la valeur -2 pour `@@FETCH_STATUS`.

Les mises à jour de valeurs clés effectuées hors du curseur sont semblables à la suppression de l'ancienne ligne suivie de l'insertion d'une nouvelle. La ligne comprenant les nouvelles valeurs n'est pas visible et si vous tentez d'extraire la ligne contenant l'ancienne valeur, le système renvoie la valeur -2 pour `@@FETCH_STATUS`. Les nouvelles valeurs sont visibles si la mise à jour s'effectue via le curseur en spécifiant la clause `WHERE CURRENT OF`.

DYNAMIC

Définit un curseur qui reflète toutes les modifications de données apportées aux lignes dans son jeu de résultats lorsque vous faites défiler le curseur. Les valeurs de données, l'ordre et l'appartenance aux lignes peuvent changer à chaque extraction.

7.1.4 Verrouillages des lignes

READ_ONLY

Interdit les mises à jour par l'intermédiaire de ce curseur. Le curseur ne peut être référencé dans une clause WHERE CURRENT OF dans une instruction UPDATE ou DELETE. Cette option supprime la fonction implicite de mise à jour d'un curseur.

SCROLL_LOCKS

Spécifie que les mises à jour ou les suppressions positionnées, effectuées par l'intermédiaire du curseur sont sûres de réussir. Les lignes sont verrouillées dès qu'elles sont lues par le curseur, de manière à garantir leur disponibilité pour des modifications ultérieures.

OPTIMISTIC

Spécifie que les mises à jour ou les suppressions positionnées, effectuées par l'intermédiaire du curseur, échouent si la ligne a été mise à jour depuis qu'elle a été lue par le curseur. SQL Server ne verrouille pas les lignes quand elles sont lues par le curseur. Au contraire, il compare les valeurs de la colonne TIMESTAMP, ou une valeur de CHECKSUM si la table ne comprend pas de colonne TIMESTAMP, afin de déterminer si la ligne a été modifiée après avoir été lue par le curseur. Si la ligne a été modifiée, la mise à jour ou la suppression positionnée que vous avez tentée échoue.

7.1.5 Génération du jeu de résultats

Le jeu de résultats est généré à partir de l'instruction SELECT spécifiée lors de la définition du curseur.

7.1.6 Opérations de mise à jour

Il est possible de préciser à ce niveau quelles sont les colonnes qui pourront faire l'objet d'une mise à jour.

UPDATE [OF Colonne 1 [...Colonne n]]

Définit les colonnes qui peuvent être mises à jour par le curseur.

Si vous indiquez UPDATE sans préciser de liste de colonnes, toutes les colonnes peuvent être mises à jour.

Si vous spécifiez OF Colonne1 [...Colonne n], seules les colonnes énumérées dans la liste peuvent être modifiées.

7.2 OUVERTURE ET LECTURE

L'instruction OPEN déclenche l'exécution de la requête SELECT sous-jacente et charge le jeu de résultats.

7.2.1 Lecture

L'instruction FETCH charge les valeurs de la ligne dans la liste de variables précisée en complément au niveau du mot clé INTO.

L'instruction **FETCH** peut être utilisée avec les arguments suivants :

NEXT

Renvoie la ligne de résultats immédiatement après la ligne courante, et incrémente cette dernière de la ligne renvoyée. Si FETCH NEXT est la première extraction effectuée sur un curseur, cette instruction renvoie la première ligne dans le jeu de résultats. NEXT est l'option d'extraction du curseur par défaut.

PRIOR

Renvoie la ligne de résultats immédiatement avant la ligne courante, et décrémente cette dernière en fonction de la ligne renvoyée. Si FETCH PRIOR est la première extraction effectuée sur un curseur, aucune ligne n'est renvoyée et le curseur reste placé avant la première ligne.

FIRST

Renvoie la première ligne dans le curseur et la transforme en ligne courante.

LAST

Renvoie la dernière ligne dans le curseur et la transforme en ligne courante.

ABSOLUTE {n | @nvar}

Si n ou @nvar est un nombre positif, cela renvoie l'énème ligne depuis le début du curseur et transforme la ligne renvoyée en nouvelle ligne courante. Si n ou @nvar est un nombre négatif, cela renvoie l'énème ligne avant la fin du curseur et transforme la ligne renvoyée en nouvelle ligne courante. Si n ou @nvar est égal à 0, aucune ligne n'est renvoyée. n doit correspondre à une valeur constante de type entier et @nvar doit être de type smallint, tinyint ou int.

RELATIVE {n | @nvar}

Si n ou @nvar est un nombre positif, cela renvoie l'énème ligne à partir de la ligne courante et transforme la ligne renvoyée en nouvelle ligne courante. Si n ou @nvar est un nombre négatif, il renvoie l'énème ligne avant la ligne courante et transforme la ligne renvoyée en nouvelle ligne courante. Si n ou @nvar est égal à 0, la ligne courante est renvoyée.

INTO

Permet de préciser la liste des variables recevant la valeur des colonnes.

Cette liste doit respecter l'ordre des colonnes de l'ordre SELECT à l'origine du jeu de résultats.

@@FETCH_STATUS

Renvoie l'état de la dernière instruction FETCH effectuée sur un curseur actuellement ouvert par la connexion.

Valeurs renvoyées par l'instruction FETCH

- 0 : L'instruction FETCH a réussi.
- 1 : L'instruction FETCH a échoué ou la ligne se situait au-delà du jeu de résultats.
- 2 : La ligne recherchée est manquante.

7.3 UN EXEMPLE A LA LOUPE

Afin de mieux comprendre la syntaxe relative au curseur, je vous propose d'étudier l'exemple suivant qui permet d'affecter des droits sur des objets de la base de données.

7.3.1 Déclaration

```
CREATE PROCEDURE DefinirDroits
    (@Operation          sysname
    ,@User               sysname
    ,@Droit              sysname
    ,@typeObjet          char(2))
AS
DECLARE @NomObjet sysname, @REQSQL varchar(255)

DECLARE Liste_Objets CURSOR
LOCAL -- Uniquement accessible dans cette procédure
FORWARD_ONLY -- Déplacement vers l'avant uniquement
READ_ONLY -- En lecture seule

FOR -- Instruction source du jeu de résultats
SELECT name
FROM sysobjects
WHERE category = '0' and type = @typeObjet
```

Figure 1 : Déclaration du curseur

7.3.2 Processus de parcours du curseur

```
OPEN Liste_Objets -- Ouverture du curseur

FETCH NEXT FROM Liste_Objets -- Lecture de la ligne suivante
INTO @NomObjet -- Chargement dans la variable @NomObjet

WHILE @@FETCH_STATUS = 0 -- Itérer tant que pas fin de curseur
BEGIN -- Bloc Instructions exécuté
    SET @REQSQL = @Operation + ' ' + @Droit + ' ON [' + @NomObjet + '] TO [' + @user + ']'
    PRINT @REQSQL
    EXEC (@REQSQL)
    FETCH NEXT FROM Liste_Objets INTO @NomObjet
END
```

Figure 2 : Parcours et traitement du jeu de résultats

Programmation SQL serveur - Déclencheurs

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

Notez : La construction dynamique de la chaîne SQL @RESQL, dont la valeur sera exécutée avec l'instruction EXEC. Cette approche s'avère très utile dès lors que l'instruction SQL n'accepte pas d'arguments sous formes de variables : ici une instruction de définition de droits (GRANT, REVOKE ou DENY).

7.3.3 Fermeture et libération mémoire

```
CLOSE Liste_Objets -- Fermeture du curseur  
DEALLOCATE Liste_Objets -- Libération de la mémoire
```

Figure 3 : Fermeture du curseur et destruction

7.4 PROCESSUS DES CURSEURS EN BREF

Pour utiliser un curseur, vous devez mettre en œuvre le processus suivant :

- Associez un curseur au jeu de résultats d'une instruction SELECT et définissez les caractéristiques du curseur, en indiquant par exemple, si les lignes contenues dans le curseur peuvent être mises à jour.
- Exécutez l'instruction OPEN pour remplir le curseur.
- Dans le curseur que vous voulez parcourir, extrayez les lignes au moyen de la commande FETCH. L'opération consistant à récupérer une ligne ou un bloc de lignes à partir d'un curseur est appelée une extraction. Le défilement est l'opération consistant à effectuer une série d'extractions afin d'extraire des lignes vers l'avant ou vers l'arrière.
- Vous pouvez éventuellement effectuer des opérations de modification sur la ligne à la position actuelle du curseur.
- Fermez le curseur à l'aide de l'instruction CLOSE.
- Libérez la mémoire allouée au curseur avec DEALLOCATE

CREDITS

ŒUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

B. Hézard – Formateur

E. Cattaneo – Formatrice

V. Bost - Formateur

Ch. Perrachon – Ingénieure de formation

Date de mise à jour : 08/02/16

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Programmation SQL serveur - Déclencheurs

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »