

Secteur Tertiaire Informatique Filière étude - développement

Activité « Développer la persistance des données »

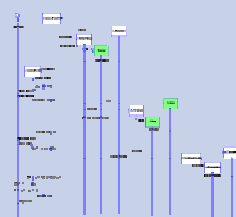
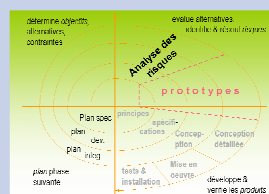
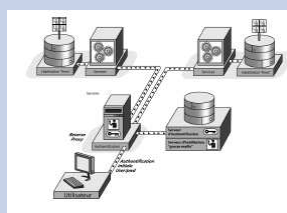
Mise en œuvre des déclencheurs DML

Accueil

Apprentissage

Période en
entreprise

Evaluation



Code barre

SOMMAIRE

I	UTILISATION DES DECLENCHEURS DML	5
II	CREATION DE DECLENCHEURS	7
III	PRINCIPE DE FONCTIONNEMENT	8
IV	MODIFICATION / SUPPRESSION	11
V	VISUALISATION D'INFORMATIONS	12
VI	IMBRICATION DES DECLENCHEURS	13

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.

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 TRANS même en l'absence d'instruction BEGIN TRANS explicite.

I UTILISATION DES DECLENCHEURS DML

Les déclencheurs servent à maintenir une intégrité référentielle 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 n'offrent pas les fonctionnalités nécessaires.

On pourra utiliser les déclencheurs, par exemple pour :

- **Modifier en cascade les tables liées 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.

Règles lors de l'utilisation de déclencheurs :

- 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 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 dans les déclencheurs : la table **deleted** contient les copies des lignes affectées par les instructions DELETE et UPDATE, la table **inserted** contient les copies des lignes affectées par l'instruction INSERT et UPDATE (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.

II CREATION DE DECLENCHEURS

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.

```
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 / EXTERNAL NAME nom_méthode
[ ENCRYPTION ]
[ EXECUTE AS Clause ]
```

SQL Server proscrit dans un déclencheur l'utilisation des instructions :

- CREATE, ALTER, DROP
- GRANT, REVOKE et DENY
- LOAD et RESTORE
- RECONFIGURE
- TRUNCATE TABLE
- UPDATE STATISTICS
- SELECT INTO (car elle crée une table)

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.

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 **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.

Avec les déclencheurs **AFTER**, l'instruction appelante est faite, et peut être annulée. Les déclencheurs **AFTER** ne peuvent être définis sur des vues.

AS EXTERNAL NAME

Spécifie la méthode d'une assembly à lier au trigger.

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.

(Référence Sécurité dans SQL Server)

III PRINCIPE DE FONCTIONNEMENT

Les étapes suivantes montrent comment un déclencheur **AFTER** est lancé lors d'un évènement sur la table ou il est défini.

Cas 1 : évènement 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 : évènement 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 aucunes lignes communes)
- Le déclencheur est lancé et ses instructions s'exécutent

Cas 3 : évènement 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

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
    UPDATE Article SET Art_Qte = Art_Qte - Vnt_Qte
    FROM Article INNER JOIN Inserted
    ON Art_Num. = Vnt_Art

```

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 la mise à jour d'une colonne donnée.

Exemple 2: Contrôle que le code département de la table Employés, de structure EMPLOYES (NOEMP, NOM, PRENOM, DEPT, SALAIRE) n'est pas modifié lors de l'insertion.

```

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

```

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 ainsi 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 2 : 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
```

Sous Studio Management,

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.

IV 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  
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.

V 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

VI 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 fonction **@@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.

Etablissement référent

Marseille Saint Jérôme

Equipe de conception

Elisabeth Cattaneo

Remerciements :

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 quelconques. »

Date de mise à jour 05/05/2008
afpa © Date de dépôt légal mai 08



**afpa / Direction de l'Ingénierie 13 place du Générale de Gaulle / 93108 Montreuil
Cedex
association nationale pour la formation professionnelle des
adultes
Ministère des Affaires sociales du Travail et de la
Solidarité**