Sommaire

NOTIONS EN TRANSACT SQL	
La déclaration d'une variable	
L'affectation d'une valeur à une variable	
L'affichage d'informations	
L'arrêt d'un programme	
L'utilisation des structures alternatives	
Exercices	5
Solutions	5
• L'utilisation des structures répétitives	7
• Le test de modification d'une colonne	······
• Le Branchement	7
La gestion des transactions	8
L'affichage des messages d'erreurs	9
• L'utilisation des Curseurs	9
Exercices	11
Solutions	12
• Création des procédures Stockées et des Triggers	
PROGRAMMATION DES PROCEDURES STOCKEES	43
Création et exécution d'une procédure stockée	
Programmation d'une procédure stockée Série l'accident (Parédaux Stadion) Serie l'accident (Parédaux Stadion) Seri	
Série d'exercices / Procédures Stockées	
Cryptage d'une procédure stockée	
Recompilation d'une procédure stockée	
Suppression d'une procédure stockée Malification d'une procédure stockée	
Modification d'une procédure stockée	
PROGRAMMATION DES DECLENCHEURS (TRIGGERS)	24
• Types de triggers	
• Fonctionnement des tables inserted et deleted :	
• Fonctionnement des triggers INSTEAD OF et AFTER :	
• Création et déclenchement d'un trigger	
Programmation d'un Trigger	
Suppression d'un trigger	
Modification d'un trigger	
Série d'exercices / Triggers	
Solution Série d'exercices / Triggers	
Cryptage d'un trigger	
EXERCICES SUPPLEMENTAIRES	36
LISTE DES REFERENCES	4 1
ANNEXE 1: SOLUTION SERIE D'EXERCICES / PROCEDURES	S
STOCKEES	
ANNEXE 2 · SOLUTION SERIE D'EXERCICES / TRIGGERS	16

Les procédures stockées sont des programmes créés et exécutés du côté serveur. Elles sont destinées à être appelées par un ou plusieurs clients de la base de données et sont très importantes pour plusieurs raisons :

- Elles sont pré-compilées à la création et donc l'utilisation d'une procédure stockée garantit un temps de réponse plus rapide et une meilleure performance système ;
- Elles évitent de réécrire plusieurs fois les mêmes instructions ;
- Elles soulagent les applications client en répartissant des traitements entre le client et le serveur ;
- Elles soulagent le réseau puisque seule l'instruction d'exécution de la procédure stockée sera envoyée à travers le réseau ;
- Il est possible de donner aux utilisateurs le droit d'exécuter une procédure stockée sans qu'ils aient le droit sur les objets qu'elle manipule.

Les Triggers (déclencheurs) sont un type particulier de procédures stockées qui ne peuvent être appelés explicitement par des applications mais se déclenchent automatiquement suite à des opérations d'ajout, de modification ou de suppression de données. Ils sont associés à certaines tables pour renforcer l'intégrité des données de la base de données ou pour intercepter des opérations sur les données demandées par l'utilisateur avant qu'elles ne soient définitivement appliquées.

La programmation des procédures stockées et des triggers nécessite la connaissance du langage Transact-SQL qui est une version améliorée de SQL utilisée par SQL Server.

Pour les exemples de ce cours, la base de données SQL Server GestionCom sera utilisée :

- Description : Cette base de données permet de stocker des commandes. Chaque commande concerne un ensemble d'articles.
- Structure de la base de données :



NOTIONS EN TRANSACT SQL

En plus des instructions SQL classiques, Transact-SQL met à la disposition des programmeurs un grand nombre d'instructions complémentaires pour :

La déclaration d'une variable

Syntaxe:

Naoual ABDALLAH	Page: 2	Version Octobre 2007

Declare @Nom_Variable Type_Donnée

Exemples:

- ✓ Declare @a int
- ✓ Declare @b varchar(10)

Remarque:

Par convention les noms des variables doivent toujours être précédés du symbole @

L'affectation d'une valeur à une variable

Syntaxe:

ou

```
Select @Nom_Variable = valeur
Select @Nom_Variable = (Select ...from...Where)
```

```
Set @Nom_Variable =valeur
Set @Nom_Variable = (Select ...from...Where)
```

Exemples:

- ✓ Select @a=1
 - -- Affecte la valeur 1 à la variable @a
- ✓ Select @a=(Select count(NumArt) from Article)
- -- Affecte le nombre d'articles enregistrés dans la table article à la variable @a
- ✓ Select @b='Table pour ordinateur'
 - -- Affecte la valeur 'Table pour ordinateur' à la variable @b

L'affichage d'informations

Syntaxe:

Print Elément_A_Afficher

Exemples:

Soient @a et @b des variables de type Chaîne de caractères, @c et @d des variables de type entier

- ✓ Print 'Bonjour' -- Affiche le texte Bonjour
- ✓ Print @a -- Affiche la valeur de @a
- ✓ Print @c -- Affiche la valeur de @c
- ✓ Print (a)c + (a)d -- Affiche la somme des variables (a)c et (a)d
- ✓ Print convert(varchar, @c) + @b -- Affiche la valeur de @c concaténé avec la valeur de @b mais puisque @c est de type numérique et qu'on ne peut jamais concaténer une valeur numérique avec une valeur chaîne de caractères, il faut passer par une fonction de conversion dont la syntaxe est la suivante :

Convert (Type de conversion, Valeur à convertir)

L'arrêt d'un programme

L'instruction return arrête l'exécution d'un programme sans condition

Naoual ABDALLAH	Page: 3	Version Octobre 2007

Syntaxe:

Return

L'utilisation des structures alternatives

• If...Else:

Syntaxe:

If Condition

Begin

Instructions

End

Else

Begin

Instructions

End

Remarques:

- ✓ Si une instruction Select apparaît dans la condition, il faut la mettre entre parenthèses
- ✓ Si dans la clause If ou Else il existe une seule instruction, on peut omettre le Begin et le End

Exemple:

On souhaite vérifier si le stock de l'article portant le numéro 10 a atteint son seuil minimum. Si c'est le cas afficher le message 'Rupture de stock':

Declare @QS

Declare @SM int

Select @QS = (Select QteEnStock from article Where

NumArt = 10

Select @SM = (Select SeuilMinimum from article Where

NumArt = 10

If @QS<=@SM

Print 'Rupture de stock'

Else

Print 'Stock disponible'

• Case : Permet d'affecter, selon une condition, une valeur à un champ dans une requête Select

Syntaxe:

Case

When Condition1 Then Résultat 1 When Condition2 Then Résultat 2

•••

Else Résultat N

End

Exemple:

Naoual ABDALLAH	Page: 4	Version Octobre 2007
Navuai Addallai i	raye. +	Version Octobre 2007

Afficher la liste des articles (Numéro, Désignation et prix) avec en plus une colonne Observation qui affiche 'Non Disponible' si la quantité en stock est égale à 0, 'Disponible' si la quantité en stock est supérieure au stock Minimum et 'à Commander' sinon :

Select NumArt, DesArt, PUArt, 'Observation' =
Case
When QteEnStock=0 then 'Non Disponible'
When QteEnStock>SeuilMinimum then 'Disponible'
Else 'à Commander'
End
From Article

Exercices

- 1. Ecrire un programme qui calcule le montant de la commande numéro 10 et affiche un message 'Commande Normale' ou 'Commande Spéciale' selon que le montant est inférieur ou supérieur à 100000 DH
- **2.** Ecrire un programme qui supprime l'article numéro 8 de la commande numéro 5 et met à jour le stock. Si après la suppression de cet article, la commande numéro 5 n'a plus d'articles associés, la supprimer.
- **3.** Ecrire un programme qui affiche la liste des commandes et indique pour chaque commande dans une colonne Type s'il s'agit d'une commande normale (montant <=100000 DH) ou d'une commande spéciale (montant > 100000 DH)
- **4.** A supposer que toutes les commandes ont des montants différents, écrire un programme qui stocke dans une nouvelle table temporaire les 5 meilleures commandes (ayant le montant le plus élevé) classées par montant décroissant (la table à créer aura la structure suivante : NumCom, DatCom, MontantCom)
- 5. Ecrire un programme qui :
 - ✓ Recherche le numéro de commande le plus élevé dans la table commande et l'incrémente de 1
 - ✓ Enregistre une commande avec ce numéro
 - ✓ Pour chaque article dont la quantité en stock est inférieure ou égale au seuil minimum enregistre une ligne de commande avec le numéro calculé et une quantité commandée égale au triple du seuil minimum

Solutions

1.

Declare @Montant decimal

Naoual ABDALLAH	Page: 5	Version Octobre 2007

```
Set @Montant=(Select Sum(PUArt*QteCommandee) from Commande C, Article
                A, LigneCommande LC where C.NumCom=LC.NumCom and
                LC.NumArt=A.NumArt and C.NumCom=10)
   If @Montant is null
         Begin
               Print 'Cette Commande n''existe pas ou elle n''a pas d''ingrédients'
               Return
         End
   if @Montant <=10000
         Print 'Commande Normale'
   Else
         Print 'Commande Spéciale'
2.
   Declare @Ote decimal
   Set @Qte=(select QteCommandee from LigneCommande where NumCom=5
              and NumArt=8)
   Delete from LigneCommande where NumCom=5 and NumArt=8
   Update article set QteEnStock=QteEnStock+@Qte where NumArt=8
   if not exists (select numcom from LigneCommande where NumCom=5)
         Delete from commande where NumCom=5
3.
   Select C.NumCom, DatCom, Sum(PUArt*QteCommandee), 'Type'=
     Case
         When Sum(PUArt*OteCommandee) <=10000 then 'Commande Normale'
          Else 'Commande Spéciale'
     End
   From Commande C, Article A, LigneCommande LC
   Where C.NumCom=LC.NumCom and LC.NumArt=A.NumArt
   Group by C.NumCom, DatCom
4.
   Create Table T1 (NumCom int, DatCom DateTime, MontantCom decimal)
   Insert into T1 Select Top 5 C.NumCom, DatCom, Sum(PUArt*QteCommandee) as Mt
   From Commande C, Article A, LigneCommande LC
   Where C.NumCom=LC.NumCom and LC.NumArt=A.NumArt
   Group by C.NumCom, DatCom
   Order by Mt Desc
5.
   if exists(select NumArt from article where QteEnStock<=SeuilMinimum)
   Begin
      Declare @a int
      set @a=(select max(NumCom) from commande) + 1
      insert into commande values(@a, getdate())
      insert into lignecommande Select @a, NumArt, SeuilMinimum * 3
                     From article Where QteEnStock <=SeuilMinimum
   End
```

Naoual ABDALLAH Page: 6 Version Octobre 2007

L'utilisation des structures répétitives

Syntaxe:

```
While Condition

Begin

instructions

End
```

Remarques:

- ✓ Le mot clé Break est utilisé dans une boucle While pour forcer l'arrêt de la boucle
- ✓ Le mot clé Continue est utilisé dans une boucle While pour annuler l'itération en cours et passer aux itérations suivantes (renvoyer le programme à la ligne du while)

Exemple:

✓ Tant que la moyenne des prix des articles n'a pas encore atteint 20 DH et le prix le plus élevé pour un article n'a pas encore atteint 30 DH, augmenter les prix de 10% et afficher après chaque modification effectuée la liste des articles. Une fois toutes les modifications effectuées, afficher la moyenne des prix et le prix le plus élevé :

```
While ((Select avg(puart) from article) < 20) and (select max(puart) from article) < 30)

Begin

Update article Set puart=puart+(puart*10)/100

Select * from article

End

Select avg(puart) as moyenne, max(puart) as [Prix élevé] from article
```

Le test de modification d'une colonne

L'instruction If Update renvoie une valeur true ou false pour déterminer si une colonne spécifique d'une table a été modifié par une instruction insert ou update (cette instruction est utilisée spécialement dans les déclencheurs et ne s'applique pas à une instruction Delete).

Syntaxe:

```
If Update (Nom_Colonne)
Begin
...
End
```

Exemple:

If update (numCom)
Print 'Numéro de commande modifié'

Le Branchement

L'instruction Goto renvoie l'exécution du programme vers un point spécifique repéré par une étiquette

Naoual ABDALLAH	Page: 7	Version Octobre 2007
-----------------	---------	----------------------

Syntaxe:

Goto Etiquette

Remarque : Pour créer une étiquette, il suffit d'indiquer son nom suivi de deux points (:)

Exemple

```
L'exemple précédent peut être écrit ainsi en utilisant l'instruction goto :

Declare @a decimal,@b decimal

Etiquette_1:

Set @a= (Select avg(puart) from article)

Set @b= (Select Max(puart) from article)

If @a<20 and @b<30

Begin

Update article Set puart=puart+(puart*20)/100

Select * from article

Goto Etiquette_1

End

Select avg(puart) as moyenne, max(puart) as [Prix élevé] from article
```

La gestion des transactions

Une transaction permet d'exécuter un groupe d'instructions. Si pour une raison ou une autre l'une de ces instructions n'a pas pu être exécutée, tous le groupe d'instructions est annulé (le tout ou rien) :

- ✓ Pour démarrer une transaction on utilise l'instruction Begin Tran
- ✓ Pour valider la transaction et rendre les traitements qui lui sont associés effectifs, on utilise l'instruction Commit Tran
- ✓ Pour interrompre une transaction en cours qui n'a pas encore été validée, on utilise l'instruction Rollback Tran
- ✓ Si plusieurs transactions peuvent être en cours, on peut leur attribuer des noms pour les distinguer

Syntaxe:

```
Begin Tran [Nom_Transaction]
...
If Condition
RollBack Tran [Nom_Transaction]
...
Commit Tran [Nom_Transaction]
```

Exemple :

Supposons qu'il n'existe pas de contrainte clé étrangère entre le champ NumCom de la table LigneCommande et le champ NumCom de la Commande.

On souhaite supprimer la commande numéro 5 ainsi que la liste de ces articles. Le programme serait :

Delete from Commande where NumCom=5

Naoual ABDALLAH Page: 8 Version Octobre 2007
--

Delete from LigneCommande where NumCom=5

Mais si, juste après l'exécution de la première instruction et alors que la deuxième n'a pas encore eu lieu, un problème survient (une coupure de courant par exemple) la base de données deviendra incohérente car on aura des lignes de commande pour une commande qui n'existe pas.

En présence d'une transaction, le programme n'ayant pas atteint l'instruction Commit Tran, aurait annulé toutes les instructions depuis Begin Tran. Le programme devra être alors :

Begin Tran

Delete from Commande where NumCom=5

Delete from LigneCommande where NumCom=5

Commit Tran

L'affichage des messages d'erreurs

L'instruction **Raiserror** affiche un message d'erreur système. Ce message est créé par l'utilisateur ou appelé à partir de la table SysMessages de la base de données Master (table contenant la liste des messages systèmes disponibles en SQL Server).

Syntaxe:

Raiserror (Num message | Texte message, gravité, état[, Param1, Param2...])

Description:

- ✓ Numéro du message : Indiquer le numéro de message pour faire appel à un message déjà disponible dans la table SysMessages.
- ✓ Texte Message : Représente le texte du message. Pour rendre certaines parties du message paramétrables, Il faut la représenter avec %d. Les valeurs à affecter à ces paramètres seront spécifiés par l'instruction raiserror (au maximum 20 paramètres peuvent être utilisées dans un message).
- ✓ Gravité : Représente le niveau de gravité. Seul l'administrateur système peut ajouter des messages avec un niveau de gravité compris entre 19 et 25 (consulter l'aide Transact-SQL dans l'analyseur de requêtes SQL pour le détail des niveaux de gravité)
- ✓ Etat : Valeur entière comprise entre 1 et 127 qui identifie la source à partir de laquelle l'erreur a été émise (consulter l'aide Transact-SQL pour le détail sur les différents états)
- ✓ Param : Paramètres servant à la substitution des variables définies dans le message. Les paramètres ne peuvent être que de type int, varchar, binary ou varbinary

L'utilisation des Curseurs

Un curseur est un groupe d'enregistrements résultat de l'interrogation d'une base de données. L'intérêt d'utiliser des curseurs est de pouvoir faire des traitements ligne par ligne chose qui n'est pas possible avec une requête SQL simple où un seul traitement sera appliqué à toutes les lignes répondant à cette requête et seul le résultat final sera visible.

Il existe plusieurs types de curseurs :

Naoual ABDALLAH	Page : 9	Version Octobre 2007
-----------------	----------	----------------------

- Curseurs à défilement en avant (Forward Only): A l'ouverture du curseur, la base de données est interrogée et la requête associée au curseur est traitée. Une copie des enregistrements répondant aux critères demandés est créée. De ce fait toutes les modifications effectuées sur les enregistrements du curseur ne seront pas visibles sur la base de données source tant que le curseur n'a pas été fermé. De même si d'autres utilisateurs ont opéré des modifications sur la base de données source, celles ci ne seront visibles que si le curseur a été fermé et ré ouvert. Ce type de curseur ne met, à la disposition de l'utilisateur, q'une seule ligne à la fois. Cette ligne peut être lue et mise à jour et l'utilisateur ne peut se déplacer que vers la ligne suivante (accès séquentiel)
- ✓ Curseurs statiques (Static): Ce curseur crée une copie statique de toutes les lignes concernées de la base de données source. Les modifications apportées ne vont être visibles que si le curseur a été fermé et ré-ouvert. L'avantage de ce type de curseur par rapport au précédent c'est que l'accès peut se faire à partir d'une ligne dans différents sens (MoveFirst, MoveNext, MovePrior, MoveLast)
- ✓ Curseurs d'ensemble de valeurs clés (Keyset): Une clé (un signet) faisant référence à la ligne d'origine de la base de données source est créée et enregistrée pour chaque ligne du curseur cela permet d'accéder aux données en temps réel à la lecture ou à la manipulation d'une ligne du curseur. Le déplacement entre les lignes dans ce genre de curseur est sans restriction (MoveFirst, MoveNext, MovePrior, MoveLast) et la mise à jour des données est possible

Remarque : La liste des membres est figée dès que l'ensemble des valeurs clés est rempli

✓ *Curseurs dynamiques (Dynamic)*: Avec ce type de curseurs, le système vérifie en permanence si toutes les lignes vérifiant la requête du curseur sont incluses. Ce curseur ne crée pas de clé sur les lignes ce qui le rend plus rapide que le curseur Keyset mais il consomme plus de ressources système.

Syntaxe

✓ Pour déclarer un curseur

✓ Pour ouvrir un curseur

Open nom_curseur

✓ Pour lire un enregistrement à partir d'un curseur

Atteindre le premier enregistrement du curseur Fetch First from nom_curseur into variable1, variable2,... Atteindre l'enregistrement du curseur suivant celui en cours

Naoual ABDALLAH Page: 10 Version Octobre 2007

Fetch Next from nom_curseur into variable1, variable2,...

ou

Fetch nom_curseur into variable1, variable2...

Atteindre l'enregistrement du curseur précédent celui en cours

Fetch Prior from nom_curseur into variable1, variable2,...

Atteindre le dernier enregistrement du curseur

Fetch Last from nom_curseur into variable1, variable2,...

Atteindre l'enregistrement se trouvant à la position n dans le curseur Fetch absolute n from nom_curseur into variable1, variable2,...

Atteindre l'enregistrement se trouvant après n positions de la ligne en cours

Fetch Relative Num_Ligne from nom_curseur into variable1, variable2,...

Remarque : La variable système @@fetch_status est utilisée pour détecter la fin du curseur. Tant que cette variable a la valeur 0, on a pas encore atteint la fin du curseur.

✓ Fermer un curseur

Close nom_curseur

✓ Libérer les ressources utilisées par un curseur :

Deallocate Nom_Curseur

Exemple:

Pour afficher la liste des articles sous la forme :

L'article Numéro ... portant la désignationcoûte

Declare @a int, @b Varchar(10), @c real

Declare Cur_ListeArt Cursor for Select NumArt, DesArt, puart from article

Open Cur_ListeArt

Fetch Next from Cur_ListeArt into @a,@b,@c

While @@fetch_status=0

Begin

Print 'L''article numéro ' + convert(varchar,@a) + ' portant la désignation ' + @b+ ' coûte ' + convert(varchar,@c)

Fetch Next from Cur_ListeArt into @a,@b,@c

End

Close Cur_ListeArt

Deallocate Cur ListeArt

Exercices

- 1. Ecrire un programme qui pour chaque commande :
 - ✓ Affiche le numéro et la date de commande sous la forme :

Commande N°: Effectuée le :

- ✓ La liste des articles associés
- ✓ Le montant de cette commande

Naoual ABDALLAH	Page : 11	Version Octobre 2007
Navuai ADDALLAII	raye . II	VEISION OCLODIC 2007

2. Ecrire un programme qui pour chaque commande vérifie si cette commande a au moins un article. Si c'est le cas affiche son numéro et la liste de ses articles sinon affiche un message d'erreur 'Aucun article pour la commande Elle sera supprimée et supprime cette commande

Solutions

```
1.
   Declare @a int, @b DateTime, @c decimal
   Declare C1 Cursor for Select C.NumCom, DatCom, Sum(PUArt*QteCommandee)
                           From Commande C, Article A, LigneCommande LC
                           Where C.NumCom=LC.NumCom and
                                 LC.NumArt=A.NumArt group by
                                 C.NumCom, DatCom
   Open C1
   Fetch Next from C1 into @a,@b,@c
   While @@fetch status =0
         Begin
               Print 'Commande N°: ' + convert(varchar,@a) + ' effectuée le: ' +
                                    convert(varchar, @b)
               Select Numart from LigneCommande where numcom=@a
               Print 'Son montant est: ' + convert(varchar,@c)
               Fetch Next from C1 into @a,@b,@c
         End
   Close C1
   Deallocate C1
2.
   Declare @a int
   Declare Cur Com Cursor for select NumCom from Commande
   open Cur_Com
   Fetch Next from Cur Com into @a
   While @@fetch_status =0
         Begin
               if not exists (Select NumArt from LigneCommande
                          where NumCom = (aa)
                     Begin
                           Print 'Aucun article pour la commande N°: '+
                                  convert(varchar, @a) + '. Elle sera supprimée'
                           Delete From Commande Where NumCom=@a
                     End
               Else
                     Begin
                           Print 'Commande n°: ' + convert(varchar, @a)
                            Select A.NumArt, DesArt, PUArt, QteCommandee
                                  From Article A, Lignecommande LC
                               Where A.NumArt=LC.NumArt and NumCom=@a
```

End
Fetch Next from Cur_Com into @a

End Close Cur_Com Deallocate Cur_Com

Création des procédures Stockées et des Triggers

Transact-SQL offre les instructions Create Procedure et Create Trigger pour la création des procédures stockées et des Triggers. Ces deux instructions seront traitées en détail dans les parties qui suivent.

PROGRAMMATION DES PROCEDURES STOCKEES

Création et exécution d'une procédure stockée

Avant de voir comment programmer une procédure stockée, Il a été jugé intéressant de voir d'abord où créer et où exécuter cette procédure stockée ainsi l'utilisateur pourra au fur et à mesure de ce cours tester en pratique l'ensemble des notions acquises.

Création d'une procédure stockée :

Il existe plusieurs manières de créer une procédure stockée :

- A partir d'entreprise Manager :
 - ✓ Accéder à la base de données concernée
 - ✓ Cliquer avec le bouton droit de la souris sur 'Procédures Stockées'
 - ✓ Choisir 'Nouvelle Procédure Stockée'



- ✓ Dans la fenêtre qui apparaît programmer la procédure créée
- ✓ Valider par le bouton OK, le système vérifie les erreurs de syntaxe et vous demande leur correction et ne pourra créer la procédure que si toutes les erreurs ont été corrigées
- A partir d'une application client : Une procédure stockée peut être créée à partir de n'importe quelle application client en respectant les contraintes et la syntaxe de cette application. Ceci dit dans le cadre de ce cours, l'utilisateur pourra utiliser en tant qu'outils client 'l'analyseur de requête SQL' livré avec SQL Server :

Naoual ABDALLAH Page: 13 Version Octobre 2007

- ✓ Ouvrir l'analyseur de requête SQL
- ✓ Se connecter au serveur si ce n'est déjà fait
- ✓ Accéder à la base de données concernée
- ✓ Programmer la procédure stockée
- ✓ Valider en cliquant sur le bouton d'exécution ou en appuyant sur la touche F5. Le système vérifie les erreurs de syntaxe et vous demande leur correction et ne pourra créer la procédure que si toutes les erreurs ont été corrigées

Remarques:

- ✓ Une procédure stockée est également appelée procédure cataloguée ou procédure mémorisée ;
- ✓ Une procédure stockée peut appeler d'autres procédures stockées (jusqu'à 32 niveaux d'imbrication en SQL Server 2000) ;
- ✓ Le système ne peut pas corriger les erreurs de logique. Il ne corrige que certaines erreurs de syntaxe. Il faut donc procéder à des jeux de test très approfondis pour corriger toutes les erreurs ;
- ✓ Une fois validée le nom de la procédure stockée apparaît dans la liste des objets procédures stockée de la base de données ;
- ✓ Il est déconseillé que le nom d'une procédure stockée commence par sp_ puisqu'au moment de l'exécution, le système recherche d'abord cette procédure parmi les procédures stockées système ce qui augmente le temps de réponse ;
- ✓ Si un utilisateur crée une procédure stockée portant le même nom qu'une procédure stockée système, cette procédure ne s'exécutera jamais
- ✓ Il est possible de regrouper plusieurs procédures de manière à pouvoir les supprimer en une seule instruction (drop nom_groupe). Pour cela le nom de la procédure doit être attribué comme suit :
- Create procedure Nom_Groupe_Procédures; Numéro_Identificateur
 ✓ Il est possible de créer des procédures stockées temporaires. Ces procédures peuvent être locales (spécifique à une connexion donnée) ou globales (reconnues au niveau de toutes les connexions jusqu'à déconnection de l'utilisateur). Pour créer une procédure temporaire locale, on lui attribut un nom qui commence par le symbole # et pour créer une procédure temporaire globale, on lui attribut un nom qui commence par ##.

Exécution d'une procédure stockée :

Les procédures stockées étant destinées à être utilisées par des applications clientes il n'existe pas de moyens pour les exécuter et voir le résultat retournés sans passer par une application Cliente. Dans le cadre de ce cours, l'utilisateur utilisera l'analyseur de requête SQL pour tester les procédures stockées créés.

Pour exécuter une procédure stockée les instructions execute et exec (format réduit de execute) peuvent être utilisées.

Programmation d'une procédure stockée

Naoual ABDALLAH	Page : 14	Version Octobre 2007

La programmation d'une procédure stockée diffère selon que cette procédure ne reçoit aucun paramètre, reçoit des paramètres en entrée, renvoie des paramètres de sortie ou retourne une valeur.

Remarques:

- ✓ Le corps de la procédure créée peut inclure n'importe quelles instructions excepté Create Procedure, Create trigger, Create View, Create Rule et Create Default;
- ✓ Une procédure stockée peut utiliser des tables temporaires locales (tables dont le nom commence par # qui sont créé dans la procédure et ne sont reconnues que dans cette procédure et dans les procédures appelant cette procédure) et des tables temporaires globales (dont le nom commence par ##).
- Sans paramètres : La procédure stockée exécute un traitement donné mais ce traitement ne dépend d'aucune valeur provenant de l'application appelante.

Syntaxe:

Create Procedure Nom_Propriétaire.Nom_Procédure as Instructions

Remarque:

Si le nom du propriétaire n'est pas spécifié (Create Procedure Nom_Procédure as...), la procédure appartient par défaut au propriétaire de la base de données (utilisateur dbo).

Exécution:

Exec Nom_Procedure

Exemples:

✓ Créer une procédure stockée nommée SP_Articles qui affiche la liste des articles avec pour chaque article le numéro et la désignation :

Create Procedure SP_Articles as

Select NumArt, DesArt from Article

✓ Exécuter cette procédure :

Exec SP_Articles

✓ Créer une procédure stockée qui calcule le nombre d'articles par commande :

Create Procedure SP_NbrArticlesParCommande as

Select Commande.NumCom, DatCom, Count(NumArt)

From Commande, LigneCommande

Where Commande.NumCom=LigneCommande.NumCom

Group by Commande.NumCom, DatCom

✓ Exécuter cette procédure :

Exec SP_NbrArticlesParCommande

• Avec des paramètres en entrée : La procédure stockée en fonction de valeurs provenant de l'extérieur va effectuer certains traitements et donc il n'est pas normal qu'une procédure stockée reçoive des paramètres en entrée dont les

Naoual ABDALLAH	Page : 15	Version Octobre 2007

valeurs ne soient pas exploitées dans les instructions des procédures (dans des tests, dans des conditions...)

Syntaxe:

Create Procedure Nom_Propriétaire.Nom_Procedure
Nom_Param1_Entrée Type_Donnée = Valeur_Par_Defaut,
Nom_Param2_Entrée Type_Donnée = Valeur_Par_Defaut...

as

Instructions

Exécution :

Exec Nom_Procedure Valeur_Param1, Valeur_Param2...

Ou

Exec Nom_Procedure Nom_Param1 = Valeur_Param1, Nom_Param2 = Valeur_Param2...

Remarque:

Avec la deuxième syntaxe, l'utilisateur n'est pas obligé de passer les paramètres dans l'ordre et en plus si des paramètres ont des valeurs par défaut, il n'est pas obligé de les passer.

Exemples:

✓ Créer une procédure stockée nommée SP_ListeArticles qui affiche la liste des articles d'une commande dont le numéro est donné en paramètre :

Create Procedure SP_ListeArticles @NumCom int as Select A.NumArt, NomArt, PUArt, QteCommandee From Article A, LigneCommande LC

Where LC.NumArt=A.NumArt and LC.NumCom=@NumCom

✓ Exécuter cette procédure pour afficher la liste des articles de la commande numéro 1 :

Exec SP ListeArticles 1

Ou

Declare @nc int

Set @nc=1

Exec SP_ListeArticles @nc

✓ Créer une procédure stockée nommée SP_ComPeriode qui affiche la liste des commandes effectuées entre deux dates données en paramètre :

Create Procedure SP_ComPeriode @DateD DateTime, @DateF DateTime as

Select * from Commande Where datcom between @dateD and @DateF

✓ Exécuter cette procédure pour afficher la liste des commandes effectuées entre le 10/10/2006 et le 14/12/2006 :

Exec SP_ComPeriode '10/10/2006', '14/12/2006'

Ou

Declare @dd DateTime, @df DateTime

Naoual ABDALLAH Page : 16 Version Octobre 2007

```
Set @dd='10/10/2006'
Set @df='14/12/2006'
Exec SP_ComPeriode @dd, @df
```

✓ Créer une procédure stockée nommée SP_TypeComPeriode qui affiche la liste des commandes effectuées entre deux dates passées en paramètres. En plus si le nombre de ces commandes est supérieur à 100, afficher 'Période rouge'. Si le nombre de ces commandes est entre 50 et 100 afficher 'Période jaune' sinon afficher 'Période blanche' (exploiter la procédure précédente) :

Exec SP_ComPeriode @DateD, @DateF

Declare @nbr int

Set @nbr=(Select count(NumCom) from Commande Where datcom between @dateD and @DateF)

If @nbr >100

Print 'Période Rouge'

Else

Begin

If @nbr<50

Print 'Période blanche'

Else

Print 'Période Jaune'

End

- ✓ Créer une procédure stockée nommée SP_EnregistrerLigneCom qui reçoit un numéro de commande, un numéro d'article et la quantité commandée :
 - Si l'article n'existe pas ou si la quantité demandée n'est pas disponible afficher un message d'erreur
 - Si la commande introduite en paramètre n'existe pas, la créer
 - Ajoute ensuite la ligne de commande et met le stock à jour

Create Procedure SP_EnregistrerLigneCom @numCom int, @numart int, @qte decimal AS

if not exists(select numart from article where numart=@numart) or (select Qteenstock from article where numart=@numart) < @qte

Begin

Print 'Cet article n"existe pas ou stock est insuffisant' Return

End

Begin transaction

if not exists(select numcom from Commande where numCom=@numcom)

insert into values(@NumCom,getdate())

commande

Naoual ABDALLAH Page: 17 Version Octobre 2007

insert into ligneCommande values(@NumCom, @Numart,@Qte)
update article set QteEnStock=QteEnStock- @Qte where
NumArt=@NumArt

Commit Transaction

• Avec des paramètres en sortie : La procédure stockée suite à un traitement réalisé va attribuer des valeurs à des paramètres en sortie. Les valeurs de ces paramètres peuvent être récupérées par des applications clientes. Il n'est pas normal qu'une procédure stockée contenant des paramètres de sortie n'affecte pas de valeurs à ces paramètres avant la fin du traitement.

Remarque : Les procédures utilisant des paramètres de sortie peuvent avoir ou ne pas avoir (selon le besoin) des paramètres en entrée

Syntaxe:

Create Procedure Nom_Propriétaire.Nom_Procedure

Nom_Param1_Entrée Type_Donnée = Valeur_Par_Defaut,

Nom_Param1_Entrée Type_Donnée = ValeurParDefaut,...

Nom_Param1_Sortie Type_Donnée Output,

Nom_Param2_Sortie Type_Donnée Output...

as

Instructions

Exécution:

Declare Var_Param1_Sortie Type_Param1_Sortie Declare Var_Param2_Sortie Type_Param2_Sortie

•••

Exec Nom_Procedure Val_Param1_Entrée, Val_Param2_Entrée..., Var_Param1_Sortie Output, Var_Param2_Sortie Output...

Exemples:

✓ Créer une procédure stockée nommée SP_NbrCommandes qui retourne le nombre de commandes :

Create Procedure SP_NbrCommandes @Nbr int output as Set @Nbr = (Select count(NumCom) from Commande)

✓ Exécuter cette procédure pour afficher le nombre de commandes :

Declare @n int

Exec SP_NbrCommandes @n Output

Print 'Le nombre de commandes : ' + convert(varchar,@n)

✓ Créer une procédure stockée nommée SP_NbrArtCom qui retourne le nombre d'articles d'une commande dont le numéro est donné en paramètre :

Create Procedure SP_NbrArtCom @NumCom int, @Nbr int output as

Set@Nbr = (Select count(NumArt) from LigneCommande where NumCom=@NumCom)

✓ Exécuter cette procédure pour afficher le nombre d'articles de la commande numéro 1 :

Declare @n int

Naoual ABDALLAH Page : 18 Version Octobre 2007

```
Exec SP_NbrArtCom 1, @n Output
Print 'Le nombre d'articles de la commande numéro 1 est : ' +
convert(varchar,@n)

Ou
Declare @nc int, @n int
Set @nc=1
Exec SP_NbrArtCom @nc, @n Output
Print 'Le nombre d'articles de la commande numéro ' +
convert(varchar,@nc) + ' est : ' +
convert(varchar,@n)
```

✓ Créer une procédure stockée nommée SP_TypePeriode qui retourne le type de la période en fonction du nombre de commande. Si le nombre de commandes est supérieur à 100, le type sera 'Période rouge'. Si le nombre de commandes est entre 50 et 100 le type sera 'Période jaune' sinon le type sera 'Période blanche' (exploiter la procédure SP_NbrCommandes) :

Create Procedure SP_TypePeriode @TypePer varchar(50)
output as

Declare @NbrCom int

Exec SP_NbrCommandes @NbrCom output

If @NbrCom >100
Set @Type='Période Rouge'

Else
Begin
If @NbrCom <50
Set @Type= 'Période blanche'

Else Set @Type= 'Période Jaune' End

• Avec valeur de retour : L'instruction return arrête l'exécution d'une procédure stockée. Une valeur entière représentant, en général, l'état d'exécution d'une procédure peut être associée à l'instruction return. Cette valeur peut être récupérée par le programme appelant de la procédure stockée.

Remarque:

- ✓ Les procédures ayant une valeur de retour peuvent avoir ou ne pas avoir (selon le besoin) des paramètres en entrée ou des paramètres de sortie ;
- ✓ Pour détecter les erreurs système, SQL Server offre la variable globale @@ERROR qui retourne le code d'erreur déclenché par SQL Server. Si la valeur de cette variable est 0 c'est qu'aucune erreur n'a été générée.

Syntaxe:

Create Procedure Nom_Propriétaire.Nom_Procedure

as

Instructions

Naoual ABDALLAH Page : 19 Version Octobre 2007

•••

Return Valeur Sortie

Exécution:

Declare Var_Retour Type_Var_Retour

•••

Exec Var_Retour=Nom_Procedure ...

Exemples:

Créer une procédure stockée nommée SP_TypePeriode qui renvoie un code de retour. Si le nombre de commandes est supérieur à 100, la procédure renvoie 1. Si le nombre de commandes est entre 50 et 100, la procédure renvoie 2. Si le nombre de commandes est inférieur à 50, la procédure renvoie 3. Si une erreur système a lieu, la procédure renvoie 4:

Create Procedure SP_TypePeriode as

Declare @NbrCom int

Set @NbrCom = (Select count(NumCom) from Commande)

If @NbrCom >=100

Return 1

If @NbrCom > 50

Return 2

If @NbrCom <=50

Return 3

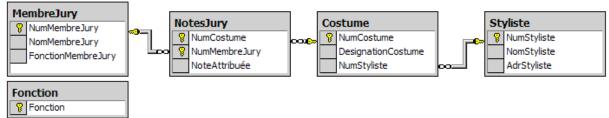
If @@ERROR <>0

Return 4

Série d'exercices / Procédures Stockées

Exercice 1

"Inter Défilés" est une société d'organisation de défilés de modes. Une de ces activités les plus réputées : Grand Défilé "Tradition Marocaine". Dans ce défilé, des costumes défilent devant un jury professionnel composé de plusieurs membres. Chaque membre va attribuer une note à chaque costume. La base de données a la structure suivante :



Créer les procédures stockées suivantes :

- **PS 1.** Qui affiche la liste des costumes avec pour chaque costume le numéro, la désignation, le nom et l'adresse du styliste qui l'a réalisé
- **PS 2.** Qui reçoit un numéro de costume et qui affiche la désignation, le nom et l'adresse du styliste concerné

Naoual ABDALLAH	Page: 20	Version Octobre 2007

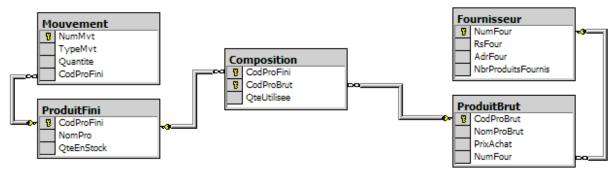
- **PS 3.** Qui reçoit un numéro de costume et qui affiche la liste des notes attribuées avec pour chaque note le numéro du membre de jury qui l'a attribué, son nom, sa fonction et la note.
- **PS 4.** Qui retourne le nombre total de costumes
- **PS 5.** Qui reçoit un numéro de costume et un numéro de membre de jury et qui **retourne** la note que ce membre a attribué à ce costume
- **PS 6.** Qui reçoit un numéro de costume et qui **retourne** sa moyenne.

Exercice 2

Une société achète à ses fournisseurs des produits bruts qu'elle utilise dans la fabrication de produits finis. On souhaite gérer la composition et les mouvements de stock de chaque produit fini.

Les Mouvements de stock sont les opérations d'entrée ou de sortie (type=S ou type=E) de produits finis vers ou depuis le magasin.

La base de données a la structure suivante :



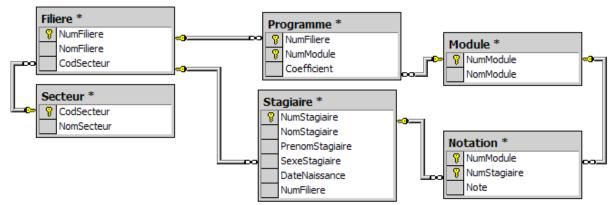
On suppose que les tables 'Mouvement', 'Produit Fini' et 'Fournisseur' sont créées. Créer les procédures suivantes :

- **PS 1.** Qui crée les tables ProduitBrut et Composition
- **PS 2.** Qui affiche le nombre de produits bruts par produit Fini
- **PS 3.** Qui retourne en sortie le prix d'achat le plus élevé
- **PS 4.** Qui affiche la liste des produits finis utilisant plus de deux produits bruts
- **PS 5.** Qui reçoit le nom d'un produit brut et **retourne** en sortie la raison sociale de son fournisseur
- **PS 6.** Qui reçoit le code d'un produit fini et qui affiche la liste des mouvements de sortie pour ce produit
- **PS 7.** Qui reçoit le code d'un produit fini et le type de mouvement et qui affiche la liste des mouvements de ce type pour ce produit fini
- **PS 8.** Qui pour chaque produit fini affiche:
 - La quantité en stock pour ce produit
 - La liste des mouvements concernant ce produit
 - La quantité totale en sortie et la quantité totale en entrée
 - La différence sera comparée à la quantité en stock. Si elle correspond afficher 'Stock Ok' sinon afficher 'Problème de Stock'
- PS 9. Qui reçoit un code produit fini et retourne en sortie son prix de reviens
- **PS 10.** Qui affiche pour chaque produit fini :
 - Le prix de reviens (utiliser la procédure précédente)
 - La liste des produits bruts le composant (nom, Mt, RSFour)
 - Le nombre de ces produits

Naoual ABDALLAH	Page : 21	Version Octobre 2007

Exercice 3

Soit la base de données suivante :



Créer les procédures stockées suivantes :

- **PS 1.** Qui affiche les numéros et les noms des stagiaires pour lesquels on a pas encore saisi de note
- **PS 2.** Qui affiche les filières ayant plus de 10 modules au programme
- **PS 3.** SP_12 qui affiche les noms des modules qui sont étudiés dans toutes les filières d'un secteur donné en paramètre
- **PS 4.** Qui affiche pour un stagiaire donné en paramètre, la liste des notes (numéro module, nom du module, note et coefficient)
- **PS 5.** Qui affiche pour chaque stagiaire :
 - Le nom, le prénom et la filière sous la forme :

Nom et Prénom :....Filière :

- S'il existe des modules où le stagiaire n'a pas de notes attribuée afficher le message 'En cours de traitement' ainsi que la liste des modules où il n'a pas encore de notes'
- S'il existe plus de deux modules où le stagiaire a obtenu une note <3 afficher 'Notes Eliminatoires' et afficher les modules concernés
- Sinon afficher la liste des modules (Module + Coefficients+note) ainsi que la moyenne du stagiaire

Solution Série d'exercices / Procédures Stockées

Voir Annexe: "Solution Série d'exercices / Procédures Stockées"

Cryptage d'une procédure stockée

Il est possible de rendre le code de la procédure stockée inaccessible, il suffit pour cela de procéder à un cryptage. La procédure stockée pourra être exécutée par un programme externe mais son contenu sera illisible que cela soit pour son propriétaire ou pour d'autres utilisateurs.

Remarque:

Cette procédure est irréversible

Syntaxe:

Create Procedure WITH ENCRYPTION as

Naoual ABDALLAH	Page: 22	Version Octobre 2007

Instructions

Recompilation d'une procédure stockée

SQL Server peut mettre en cache le plan d'exécution d'une procédure stockée. L'utilisateur peut demander la recompilation de la procédure stockée. Trois possibilités sont offertes :

• Si le concepteur souhaite que la procédure stockée soit recompilée à chaque exécution, il doit indiquer cela au moment de la création de la procédure :

Syntaxe:

Create Procedure WITH RECOMPILE as

Instructions

• Si l'utilisateur souhaite recompiler une procédure stockée, il peut le faire au moment de l'exécution de la procédure :

Syntaxe:

Exec Procedure WITH RECOMPILE

Ou alors faire appel à la procédure stockée système sp_recompile

Suppression d'une procédure stockée

Syntaxe:

Drop Procedure Nom_Procédure

Exemple:

Drop Procedure NbrArticlesCommande

Modification d'une procédure stockée

Il existe plusieurs manières de créer une procédure stockée :

- A partir d'entreprise Manager :
 - Accéder à la procédure stockée concernée, double cliquer dessus. Apporter les modifications souhaitées et valider
- A partir d'une application client : Une procédure stockée peut être modifiée en utilisant l'instruction Transact-SQL Alter Procedure à partir de n'importe quelle application client :

Syntaxe:

Alter Procedure Nom_Procédure as Nouvelles instructions

Remarque:

Le modification du contenu d'une procédure stockée n'affecte pas les permissions d'accès associées à cette procédure ce qui n'est pas le cas lors d'une suppression.

	Naoual ABDALLAH	Page : 23	Version Octobre 2007
--	-----------------	-----------	----------------------

PROGRAMMATION DES DECLENCHEURS (TRIGGERS)

Les triggers peuvent intercepter les opérations sur les données de la table avant qu'elles ne soient définitivement appliquées. Ils peuvent alors interrompre les traitements de mise à jour et selon certaines conditions annuler ces modifications, leur associer des traitements complémentaires ou laisser le système poursuivre leur validation. Les déclencheurs peuvent être associés à trois types d'actions de déclenchement sur une table :

- Déclencheurs d'insertion : Se déclenchent suite à une opération d'ajout d'enregistrements dans la table ;
- Déclencheurs de modification : Se déclenchent suite à une opération de modification des enregistrements de la table ;
- Déclencheurs de suppression : Se déclenchent suite à une opération de suppression d'enregistrements à partir de la table.

Remarque |

Les triggers consomment peu de ressources système à condition qu'ils n'utilisent pas de curseurs.

Types de triggers

Les déclencheurs peuvent être de deux types : INSTEAD OF et AFTER.

- Les déclencheurs INSTEAD OF:
 - ✓ Sont exécutés à la place de l'action de déclenchement ;
 - ✓ Sont vérifiés avant les contraintes d'intégrité associées à la table ce qui permet de mettre en place des traitements qui complètent les actions de ces contraintes ;
 - ✓ Peuvent être associés aussi bien à des tables qu'à des vues ce qui permet la mise à jour des données associées à ces vues ;
 - ✓ Ne peuvent être associés à des tables **cible** de contraintes d'intégrité référentielle en cascade ;
 - ✓ Un seul déclencheur INSTEAD OF est autorisé par action de déclenchement dans une table
 - ✓ Même si un trigger INSTEAD OF contient une action d'insertion sur la table ou la vue à laquelle il est associé, il ne sera jamais exécuté à nouveau (exécution non récursive).
- Les déclencheurs AFTER :
 - ✓ Sont exécutés après la validation des contraintes associées à la table. Si une contrainte n'est pas vérifiée ce type de déclencheurs ne se déclenchera jamais ;
 - ✓ Ne peuvent être associés qu'à des tables ;
 - ✓ Plusieurs déclencheurs AFTER sont autorisés sur une même table et pour une même action de déclenchement. La procédure stockée système **sp_SetTriggerOrder** permet de spécifier le premier et le dernier déclencheur à exécuter pour une action :

Exec sp_SetTriggerOrder

@triggername = 'MyTrigger',

Naoual ABDALLAH	Page: 24	Version Octobre 2007

@order = 'first|Last|None',
@stmttype = 'Insert|Update|Delete'

Fonctionnement des tables inserted et deleted :

Au cours des opérations d'ajout, de suppression et de modification, le système utilise les tables temporaires inserted et deleted. Ces tables ne sont accessibles qu'au niveau des triggers et leur contenu est perdu dès que les triggers sont validés.

- Action d'ajout : Les enregistrements ajoutés sont placés dans une table temporaire nommée inserted ;
- Action de suppression : Les enregistrements supprimés sont placés dans une table temporaire nommée deleted.
- Action de modification : L'opération de modification est interprétée comme une opération de suppression des anciennes informations et d'ajout des nouvelles informations. C'est pourquoi le système utilise dans ce cas les deux tables temporaires deleted et inserted. En fait quand un utilisateur demande à modifier des enregistrements, ceux ci sont d'abord sauvegardés dans la table temporaire deleted et la copie modifiée est enregistrée dans la table inserted.

Fonctionnement des triggers INSTEAD OF et AFTER :

• Cas où seul un trigger INSTEAD OF est associé à l'action de mise à jour (insert, delete ou update) : Dans le trigger INSTEAD OF, les enregistrements ajoutés (respectivement modifiés ou supprimés) apparaissent uniquement dans les tables temporaires mais pas dans la table d'origine et si le code associé à ce trigger ne prend pas en charge l'ajout (respectivement la modification ou la suppression) de ces enregistrements, ils ne seront pas ajoutés (respectivement modifiés et supprimés) même si aucune action n'annule le déclencheur.

Exemple:

Un utilisateur exécute l'action suivante :

Insert into commande values (100,'13/09/07')

Supposons qu'un trigger instead of est associé à l'action d'insertion sur la table commande. Dans le corps de ce trigger, on affiche le contenu de la table inserted et le contenu de la table commande.

Dans la table inserted, on remarquera la présence de la commande numéro 100 mais dans la table commande cet enregistrement est absent et ne sera pas ajouté à la table commande même après la fin de l'exécution de l'action d'ajout. Ceci est dû au fait que l'exécution des triggers instead of remplace l'action de déclenchement.

• Cas où seul des triggers AFTER sont associés à l'action de mise à jour (insert, delete ou update) : Les contraintes sont testées en premier. Si une contrainte n'est pas vérifiée l'insertion est annulée sans que le trigger soit exécuté. Si les contraintes sont vérifiées, le trigger est exécuté. Les enregistrements ajoutés apparaissent et dans la table d'origine et dans les

Naoual ABDALLAH Page: 25 Version Octobre 2007

- tables temporaires concernées par l'action. Si dans le code associé à ce trigger, aucune action n'annule la transaction, l'opération est validée.
- Cas où un trigger INSTEAD OF ainsi que des triggers AFTER sont associés à l'action de mise à jour (insert, delete ou update) : Le trigger INSTEAD OF est exécuté en premier, les enregistrements concernés par l'action de mise à jour (insert, delete ou update) apparaissent uniquement dans les tables temporaires mais pas dans la table d'origine et si le code associé à ce trigger ne prend pas en charge les opérations sur ces enregistrements, ils ne seront pas ajoutés (modifiés ou supprimés) même si aucune action n'annule le trigger et les triggers AFTER ne seront pas exécutés.

Si le trigger INSTEAD OF, déclenche une opération (ajout, modification ou suppression) sur la même table, les triggers AFTER vont se déclencher et les tables temporaires au sein de ces triggers vont contenir les nouvelles valeurs manipulées.

Si d'autres instructions se trouvent après l'instruction de mise à jour (insert, delete ou update) dans le trigger instead of, elles seront exécutées après la fin de l'exécution des triggers After sauf si une instruction Rollback a été rencontrée.

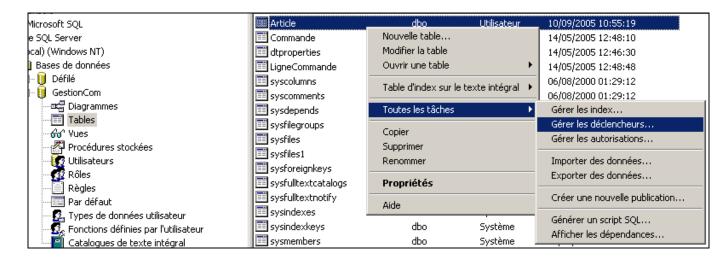
Création et déclenchement d'un trigger

Avant de voir comment programmer un trigger, Il a été jugé intéressant de voir d'abord où créer et comment déclencher un trigger ainsi l'utilisateur pourra au fur et à mesure de ce cours tester en pratique l'ensemble des notions acquises.

Création d'un trigger :

Il existe plusieurs manières de créer un trigger :

- A partir d'entreprise Manager :
 - ✓ Accéder à la base de données concernée
 - ✓ Accéder à la table concernée
 - ✓ Cliquer dessus avec le bouton droit de la souris sur l'option 'Toutes les tâches'
 - ✓ Choisir 'Gérer les déclencheurs'

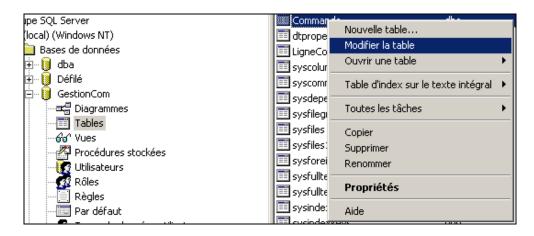


Naoual ABDALLAH Page : 26 Version Octobre 2007

- ✓ Dans la fenêtre qui apparaît programmer le trigger souhaité
- ✓ Valider par le bouton OK, le système vérifie les erreurs de syntaxe et vous demande leur correction et ne pourra créer le trigger que si toutes les erreurs ont été corrigées

Ou alors:

- ✓ Accéder à la base de données concernée
- ✓ Accéder à la table concernée
- ✓ Cliquer dessus avec le bouton droit de la souris et choisir l'option 'Modifier la table'



✓ Dans la fenêtre ci-dessous qui apparaît, appuyer sur le bouton de la barre d'outils se trouvant après le bouton indiquant la clé primaire :



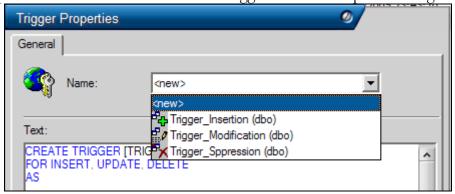
- ✓ Dans la fenêtre qui s'affiche, programmer le trigger
- ✓ Valider par le bouton OK
- A partir d'une application client : Un trigger peut être créé à partir de n'importe quelle application client en respectant les contraintes et la syntaxe de cette application. Dans le cadre de ce cours, 'l'analyseur de requête SQL' livré avec SQL Server sera utilisé pour les tests :
 - ✓ Ouvrir l'analyseur de requête SQL
 - ✓ Se connecter au serveur si ce n'est déjà fait
 - ✓ Accéder à la base de données concernée
 - ✓ Programmer le trigger
 - ✓ Valider en cliquant sur le bouton d'exécution ou en appuyant sur la touche F5. Le système vérifie les erreurs de syntaxe et vous

Naoual ABDALLAH Page: 27 Version Octobre 2007

demande leur correction et ne pourra créer le trigger que si toutes les erreurs ont été corrigées

Remarques:

✓ La liste des triggers déjà créés pour une table peut être consulté à partir de la fenêtre de création du trigger dans l'Entreprise Manager



Notez le symbole associé à chaque type de trigger;

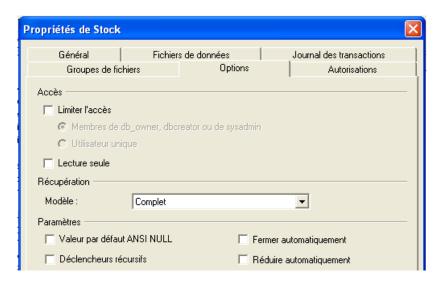
- ✓ Il n'est pas possible d'associer un trigger à une table temporaire ou à une table système ;
- ✓ L'instruction **Truncate Table Nom_Table** a le même effet que **Delete From Nom_Table** mais ne déclenche pas les triggers associés à delete (l'instruction TRUNCATE TABLE n'est pas inscrite au journal);
- ✓ Un trigger ne devrait renvoyer aucun résultat. Il ne faut donc pas que le corps d'un trigger contienne des instructions Select ou des affectations de valeurs à des variables (si des valeurs doivent être affectées à des variables, il faut utiliser l'instruction **Set NoCount** au début du déclencheur, pour empêcher le renvoi de ces valeurs);
- ✓ Si une transaction utilisateur est exécutée et que cette transaction exécute une instruction qui active un déclencheur contenant une action Rollback Transaction alors toute la transaction est annulée ;
- ✓ Si un ensemble d'instructions doivent être exécutées et qu'une de ces instructions active un déclencheur contenant une action Rollback Transaction alors toutes les instructions qui la suivent dans la liste seront également annulées ;
- ✓ La variable système @@RowCount renvoie dans un déclencheur le nombre de lignes affectées par l'instruction qui a déclenché le trigger;
- ✓ En programmant un trigger, il faut prendre en considération le fait que plusieurs enregistrements peuvent être affectés en une seule opération d'ajout, de modification ou de suppression);
- ✓ Un trigger peut exécuter une action qui déclenche un autre trigger et ce dernier trigger peut exécuter une action qui déclenche un troisième trigger et ainsi de suite. SQL Server autorise 32 niveaux d'imbrications au maximum ;

Naoual ABDALLAH Page : 28 Version Octobre 2007

✓ L'option de configuration du serveur **nested triggers** détermine si les déclencheurs imbriqués sont autorisés ou non (Si nested triggers est à 0, l'imbrication n'est pas autorisée). Cette option peut également être configurée dans la page de propriétés du serveur :



- ✓ Un trigger peut s'auto-appeler s'il inclut une instruction qui le déclenche (récursivité directe) ou si il exécute une action qui fait appel à un autre trigger qui lui inclut une instruction qui appelle le trigger d'origine (récursivité indirecte) ;
- ✓ La récursivité directe est autorisée ou non selon que l'option serveur Recursive_Triggers est à on ou à off. Cette option peut également être configurée dans la page de propriétés de la base de données :



✓ La récursivité indirecte est désactivée si l'option serveur Recursive_Triggers est à off et l'option nested triggers est à 0.

Exécution d'un trigger :

Les triggers s'exécutent automatiquement suite à des opérations de mise à jour sur les tables.

Naoual ABDALLAH Page: 29 Version Octobre 2007

- Pour tester un trigger d'ajout, il faut donc ajouter des enregistrements à la table à laquelle il est associé et vérifier les répercutions en fonction des traitements contenus dans le trigger (n'oubliez pas que certaines instructions SQL permettent d'ajouter plusieurs enregistrements en une seule fois à une table)
- Pour tester un trigger de modification, il faut donc modifier des enregistrements de la table à laquelle il est associé et vérifier les répercutions en fonction des traitements contenus dans le trigger (n'oubliez pas que plusieurs champs dans plusieurs enregistrements peuvent être modifiés en une seule fois)
- Pour tester un trigger de suppression, il faut donc supprimer des enregistrements depuis la table à laquelle il est associé et vérifier les répercutions en fonction des traitements contenus dans le trigger (n'oubliez pas que plusieurs enregistrements peuvent être supprimés en une seule fois)

Remarque:

Entreprise Manager ne permet l'ajout et la modification que d'une seul enregistrement à la fois. Pour réaliser des tests sur des triggers d'insertion et de modification, utilisez plutôt l'analyseur de requête.

Programmation d'un Trigger

Syntaxe:

Create Trigger Nom_Trigger On Nom_Table Instead Of | For Opération1, Opération2... As

Instructions

Remarque:

- ✓ Opération peut prendre Insert, Delete ou Update selon le type de trigger à créer
- ✓ Un même trigger peut être associé à une seule opération ou à plusieurs opérations à la fois
- ✓ A chaque table, peuvent être associées trois triggers au maximum : ajout, modification et suppression (un trigger concernant deux opérations est compté comme deux triggers)
- ✓ Le corps du trigger créé peut inclure n'importe quelles instructions excepté Create Database, Alter Database, Drop Database, Restore Database, Restore Log et reconfigure ;

Exemples:

✓ Le trigger suivant interdit la modification des commandes Create Trigger Tr_Empêcher_Modif On Commande For Update

As

Naoual ABDALLAH Page : 30 Version Octobre 2007

Rollback

✓ Le trigger suivant interdit la modification du numéro de commande et vérifie si la date saisie pour la date de commande est supérieure ou égale à la date du jour

```
Create Trigger Tr_Empêcher_Modif_Numcom
On Commande
For Update
As
     if update(NumCom)
      Begin
            Raiserror('le numéro de commande ne peut être
                      modifié',15,120)
            Rollback
      End
     if update(DatCom)
      Begin
            if ((select count (DatCom) from inserted
            Where datediff(day,datcom,getdate())>0>>0
                  Begin
                      Raiserror('La date de commande ne peut
                      pas être inférieur à la date en cours',15,120)
                      Rollback
                  End
      End
```

✓ Le trigger suivant empêche la suppression des commandes ayant des articles associés

Remarque : Ce trigger ne se déclenchera pas s'il existe une contrainte clé étrangère entre le champ NumCom de la table ligneCommande et le champ NumCom de la table commande.

```
Com de la table commande.

Create Trigger Tr_Empêcher_Suppr
On Commande
For Delete
As

Declare @a int
set @a = (Select count(numart) from lignecommande, deleted
where lignecommande.numcom = deleted.numcom)
if (@a>0)
Begin

Raiserror('Opération annulée. Une ou plusieurs
commandes ont des articles enregistrés',15,120)
Rollback
End
```

✓ Le trigger suivant à la suppression d'une ligne de commande, remet à jour le stock et vérifie s'il s'agit de la dernière ligne pour cette commande. Si c'est le cas la commande est supprimée :

Create Trigger Tr_Supprimer_Ligne

Naoual ABDALLAH Page: 31 Version Octobre 2007

Version Octobre 2007

Naoual ABDALLAH

```
On LigneCommande
      For Delete
      As
      Update article set
                            QteEnStock
                                              QteEnStock
                                                                (select
      Sum(QteCommandee)
                                                 deleted
                                   from
                                                                where
      article.NumArt=deleted.NumArt)
                                              article. deleted
                                                                where
                                       from
      deleted.numart=article.numart
      Delete from commande where numcom not in (select numcom
      from lignecommande)
✓ Le trigger suivant à l'ajout d'une ligne de commande vérifie si les
   quantités sont disponibles et met le stock à jour
   Create Trigger Tr_Ajouter_Ligne
   On LigneCommande
   For Insert
   As
      Declare @a int
      set @a=(select count(numart) from inserted, article where
      article.numart
                            inserted.numart
                                               and
                                                      QteCommandee
      >QteEnStock)
      if (@a > 0)
            Begin
                   Raiserror ('Ajout refusé. Quantités demandées non
                            disponibles en stock',15,120)
                   Rollback
            End
      Else
            Update article set QteEnStock = QteEnStock -
                          (select Sum(QteCommandee) from inserted
                          where article.NumArt=inserted.NumArt)
            From article, inserted where inserted.numart=article.numart
✓ Le trigger suivant à la modification d'une ligne de commande vérifie si
   les quantités sont disponibles et met le stock à jour
   Create Trigger Tr_Modifier_Ligne
   On LigneCommande
   For Update
   As
      Declare @a int
      set @a=(select count(numart) from inserted, deleted, article where
                article.numart = inserted.numart and article.numart =
                deleted.numart
                                 and
                                         inserted.QteCommandee
                QteEnStock+deleted.QteCommandee)
      if (@a > 0)
            Begin
                   Raiserror("Modification refusée. Quantités demandées
                            non disponibles en stock',15,120)
                   Rollback
            End
```

Page: 32

Else

update article set QteEnStock = QteEnStock

- + (select Sum(QteCommandee) from deleted where deleted.NumArt=Article.NumArt)
- (select Sum(QteCommandee) from inserted where inserted.NumArt=Article.NumArt)

From article, inserted, deleted where inserted.numart = article.numart and article.numart = deleted.numart

Remarque:

Si le trigger déclenché effectue une opération sur une autre table, les triggers associés à cette table sont alors déclenchés (principe de cascade)

Suppression d'un trigger

Syntaxe:

Drop Trigger Nom_Trigger

Modification d'un trigger

Syntaxe:

Alter Trigger Nom_Trigger On Nom_Table For Opération1, Opération2...

as

Nouvelles Instructions

Série d'exercices / Triggers

Exercice 1

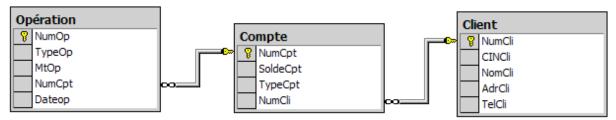
Soit la base de données Défilé citée dans la série d'exercices sur les procédures stockées. Créer les triggers suivants :

- **TR 1.** Qui, à l'ajout de costumes dans la table Costume, vérifie si les numéros de stylistes concernés existent dans la table Styliste. Si ce n'est pas le cas annuler l'opération d'ajout
- **TR 2.** Qui, à la suppression de costumes, vérifie si des notes leur ont été attribuées. Si c'est le cas empêcher la suppression
- **TR 3.** Qui, à l'affectation de notes à des costumes dans la table NotesJury, vérifie si les costumes et les membres de jury existent et si les notes attribuées sont comprises entre 0 et 20
- **TR 4.** Qui à l'ajout de membres jury, cherche si leurs fonctions existent dans la table Fonction si ce n'est pas le cas il les rajoute

Exercice 2

Soit la base de données GestionCompte suivante :

Naoual ABDALLAH	Page : 33	Version Octobre 2007



- Les opérations consistent en des opérations de retrait ou de dépôt d'argent (TypeOp=D si le client a déposé de l'argent sur son compte et TypeOp=R si le client a retiré de l'argent sur son compte)
- Un client ne peut avoir qu'un seul compte courant (TypeCpt="CC") et qu'un seul compte sur carnet (TypeCpt="CN")
- Le numéro d'opération est automatique
- La date de l'opération prend par défaut la date du jour

Créer les triggers suivants :

- **TR 1.** Qui à l'ajout ou à la modification de clients dans la table client vérifie si les Numéros de CIN saisies n'existent pas pour d'autres clients
- **TR 2.** Qui à la création de comptes, vérifie si :
 - ✓ Les soldes sont supérieurs à 1500 DH;
 - ✓ Les types de compte sont CC ou CN et aucune autre valeur n'est acceptée ;
 - ✓ Les clients n'ont pas déjà des comptes du même type.
- **TR 3.** Qui interdit la suppression de comptes dont le solde est > 0 ou de comptes pour lesquels la dernière opération date de moins de 3 mois même s'ils sont vides (solde=0).
- **TR 4.** Qui:
 - ✓ Interdit la modification des numéros de comptes ;
 - ✓ Interdit la modification du solde de comptes auxquels sont associées des opérations ;
 - ✓ Ne permet pas de faire passer des comptes sur carnet en comptes courants, le contraire étant possible ;
 - ✓ A la modification de numéros de clients vérifie si les nouveaux clients n'ont pas de comptes associés du même type.

Exercice 3

Soit la base de données stock citée dans la série d'exercices sur les procédures stockées. Créer les triggers suivants :

- **TR 1.** Qui à l'ajout de produits bruts dans la table 'Produit Brut' met à jour le champ NbrProduitsfournis pour les fournisseurs concernés
- **TR 2.** Qui à la suppression de produits bruts dans la table 'Produit Brut' met à jour le champ NbrProduitsfournis pour les fournisseurs concernés
- TR 3. Qui à l'ajout de mouvements dans la table mouvement met à jour le stock
- **TR 4.** Qui à la suppression de mouvements dans la table mouvement met à jour le stock
- **TR 5.** Qui à la modification de mouvements dans la table mouvement met à jour le stock

Naoual ABDALLAH	Page: 34	Version Octobre 2007

Exercice 4 (suite exercice 2)

Remarque : pour cette partie on éliminera la contrainte de TR4 interdisant la modification du solde d'un compte auquel sont associées des opérations

- TR 5. Qui à l'ajout d'opérations met à jour le solde client
- TR 6. Qui à la suppression d'opérations met à jour le solde client
- TR 7. Qui:
 - ✓ Empêche la modification des champs numéro, date de l'opération et numéro de compte ;
 - ✓ A la modification des Montants des opérations ou des types des opérations met à jour le solde client.

Solution Série d'exercices / Triggers

Voir Annexe: "Solution Série d'exercices / Triggers"

Cryptage d'un trigger

Comme pour une procédure stockée, il est possible de rendre le code d'un trigger inaccessible, il suffit pour cela de procéder à un cryptage. Le trigger pourra être exécuté par un programme externe mais son contenu sera illisible que cela soit pour son propriétaire ou pour d'autres utilisateurs.

Remarque:

Cette procédure est irréversible

<u>Syntaxe :</u>

Create Trigger WITH ENCRYPTION as Instructions

Naoual ABDALLAH	Page : 35	Version Octobre 2007
Navual ADDALLAII	raue: 33	VEISION OCLODIC 2007

EXERCICES SUPPLEMENTAIRES

Exercice1

Soit la base de données suivante :

- Recettes (NumRec, NomRec, MethodePreparation, TempsPreparation)
- Ingrédients (NumIng, NomIng, PUIng, UniteMesureIng, NumFou)
- Composition_Recette (<u>NumRec</u>, <u>NumIng</u>, QteUtilisee)
- Fournisseur(NumFou, RSFou, AdrFou)

Créer les procédures stockées suivantes :

- **PS 1.** Qui affiche la liste des ingrédients avec pour chaque ingrédient le numéro, le nom et la raison sociale du fournisseur
- PS 2. Qui affiche pour chaque recette le nombre d'ingrédients et le prix de reviens
- **PS 3.** Qui affiche la liste des recettes qui se composent de plus de 10 ingrédients avec pour chaque recette le numéro et le nom
- PS 4. Qui reçoit un numéro de recette et qui retourne son nom
- **PS 5.** Qui reçoit un numéro de recette. Si cette recette a au moins un ingrédient, la procédure retourne son meilleur ingrédient (celui qui a le montant le plus bas) sinon elle retourne "Aucun ingrédient associé"
- **PS 6.** Qui reçoit un numéro de recette et qui affiche la liste des ingrédients correspondant à cette recette avec pour chaque ingrédient le nom, la quantité utilisée et le montant
- PS 7. Qui reçoit un numéro de recette et qui affiche :
 - ✓ Son nom (Procédure PS_4)
 - ✓ La liste des ingrédients (procédure PS_6)
 - ✓ Son meilleur ingrédient (PS_5)
- **PS 8.** Qui reçoit un numéro de fournisseur vérifie si ce fournisseur existe. Si ce n'est pas le cas afficher le message 'Aucun fournisseur ne porte ce numéro' Sinon vérifier, s'il existe des ingrédients fournis par ce fournisseur si c'est le cas afficher la liste des ingrédients associées (numéro et nom) Sinon afficher un message 'Ce fournisseur n'a aucun ingrédient associé. Il sera supprimé' et supprimer ce fournisseur
- **PS 9.** Qui affiche pour chaque recette :
 - ✓ Un message sous la forme : "Recette : (Nom de la recette), temps de préparation : (Temps)
 - ✓ La liste des ingrédients avec pour chaque ingrédient le nom et la quantité
 - ✓ Un message sous la forme : Sa méthode de préparation est : (Méthode)
 - ✓ Si le prix de reviens pour la recette est inférieur à 50 Dh afficher le message 'Prix intéressant'

Exercice2

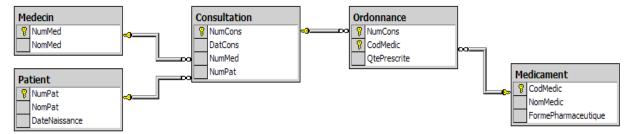
Une clinique souhaite gérer les consultations médicales effectuées. Dans cette clinique :

- Une consultation médicale est réalisée par un et un seul médecin pour un et un seul patient
- Lors d'une consultation, plusieurs médicaments peuvent être prescrits
- Plusieurs médicaments peuvent avoir le même nom mais des codes différents selon les formes pharmaceutiques disponibles (par exemple Doliprane peut exister sous

Naoual ABDALLAH	Page : 36	Version Octobre 2007

forme de comprimés, de suppositoires et effervescent. Doliprane sera enregistré 3 fois avec des codes différents)

L'analyse de ce système d'information nous a permis de dégager la base de données suivante :



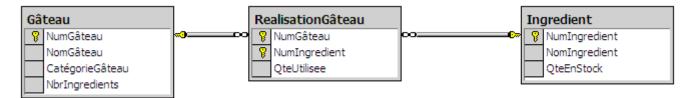
Créer les procédures stockées suivantes :

- **PS 1.** Qui affiche la liste des consultations triées par date décroissante avec pour chaque consultation le nom du médecin traitant et le nom du patient concerné
- PS 2. Qui affiche le code du médicament qui a été prescrit le plus de fois
- **PS 3.** Qui affiche le code du médicament qui a été prescrit dans toutes les consultations
- **PS 4.** Qui supprime toutes les consultations ayant plus de cinq ans
- **PS 5.** Qui reçoit un nom de médicament et qui affiche les différentes formes pharmaceutiques disponibles pour ce médicament
- PS 6. Oui retourne le nombre de médecins
- **PS 7.** Qui reçoit un numéro de consultation et un code médicament. La procédure retourne 0 si cette consultation n'existe pas ou si ce médicament n'existe pas pour cette consultation et la procédure retourne la quantité prescrite sinon
- **PS 8.** Qui reçoit un numéro de médecins et qui retourne le nombre de consultations qu'il a effectué le jour même
- **PS 9.** Qui reçoit deux dates et qui affiche la liste des consultations effectuées entre ces deux dates
- PS 10. Qui reçoit un numéro de consultation et qui affiche :
 - ✓ La date de consultation
 - ✓ Le nom du médecin
 - ✓ Le nom du patient
 - ✓ La liste des médicaments prescrits avec pour chaque médicament le Nom, la forme pharmaceutique et la quantité prescrite
- **PS 11.** Qui affiche la liste des consultations avec pour chaque consultation :
 - ✓ Le numéro
 - ✓ La date de consultation
 - ✓ Le nom du médecin
 - ✓ Le nom du patient
 - ✓ La liste des médicaments prescrits avec pour chaque médicament le Nom, la forme pharmaceutique et la quantité prescrite

Exercice 3

Dans une pâtisserie, on souhaite gérer le stock en ingrédients de base (farine, œufs...) qui entrent dans la composition des gâteaux. L'analyse de ce système d'information nous a permis de dégager la base de données suivante :

	Naoual ABDALLAH	Page : 37	Version Octobre 2007
--	-----------------	-----------	----------------------

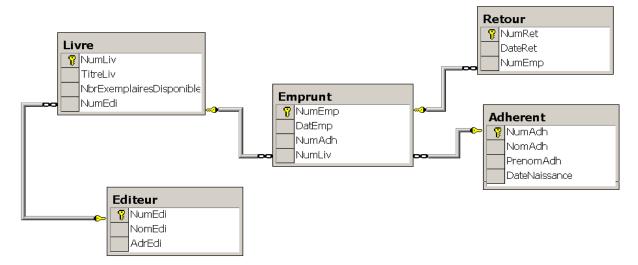


Créer les Triggers suivants :

- **TR 1.** Qui empêche la modification du numéro, du nombre d'ingrédients pour un gâteau
- TR 2. Qui vérifie que deux ingrédients ne portent pas le même nom
- **TR 3.** Qui vérifie que deux gâteaux d'une même catégorie ne portent pas le même nom
- **TR 4.** Qui ne permet pas l'enregistrement de plus de 20 gâteaux
- TR 5. Qui empêche la suppression de gâteaux ayant des ingrédients
- **TR 6.** Qui à l'enregistrement d'ingrédients pour des gâteaux, met à jour le stock d'ingrédients et met à jour le nombre d'ingrédients

Exercice 4:

Soit la base de données suivante :



Créer les procédures stockées suivantes :

- **PS 1.** Qui affiche la liste des livres qui n'ont jamais été empruntés (Numéro du livre, Titre du livre, Nom de l'éditeur)
- **PS 2.** Qui affiche la liste des livres empruntés et non encore retournés (Numéro du livre, Titre du livre, Numéro de l'adhérent, Nom de l'adhérent et date d'emprunt)
- **PS 3.** Qui affiche la liste des livres pour lesquels il existe des exemplaires disponibles (Numéro et titre du livre)
- **PS 4.** Qui affiche la liste des livres sous la forme : Livre numéro : (NumLiv), son titre est : (TitreLiv). Il été édité par : (NomEdi)
- **PS 5.** Qui reçoit le numéro d'un éditeur et qui retourne le nombre de livres qu'il a édité
- PS 6. Qui retourne le nom de l'éditeur qui a édité le plus de livres

Naoual ABDALLAH Page: 38 Version Octobre 2007

- **PS 7.** Qui reçoit un titre de livre et qui retourne la valeur 1 s'il a déjà été emprunté et la valeur 0 sinon
- PS 8. Qui reçoit un numéro de livre et un numéro d'adhérent et qui :
 - ✓ Vérifie si ce numéro de livre existe. Si ce n'est pas le cas, il affiche le message d'erreur "Ce livre n'existe pas" sous forme d'un message système ;
 - ✓ Vérifie si ce numéro d'adhérent existe. Si ce n'est pas le cas, il affiche le message d'erreur "Cet adhérent n'existe pas" sous forme d'un message système;
 - ✓ Vérifie si au moins un exemplaire est disponible de ce livre. Si ce n'est pas le cas il affiche le message d'erreur "Aucun exemplaire disponible" sous forme d'un message système;
 - ✓ Enregistre l'emprunt de ce livre pour cet adhérent (la date de l'emprunt prend la date du jour)
- **PS 9.** Qui pour chaque livre affiche:
 - ✓ Les informations sur le livre sous la forme :

Titre :

Nombre exemplaire: ...

Editeur:

- ✓ La liste des emprunts effectués (Nom Adhérent et Date d'emprunt)
- ✓ Le nombre d'exemplaires en cours d'emprunt

Créer les triggers suivants :

- **TR 1.** A l'ajout de livres, ce trigger vérifie que le nombre d'exemplaires pour ces livres est >0
- TR 2. A l'ajout d'adhérents, ce trigger vérifie si ces adhérents ont plus de 7 ans d'âge
- **TR 3.** Deux éditeurs ne peuvent pas porter le même nom et donc à l'ajout ou à la modification d'éditeurs, ce trigger doit vérifier s'il n'existe pas déjà des éditeurs portant les mêmes noms
- **TR 4.** Ce trigger doit empêcher la modification des emprunts
- **TR 5.** A l'enregistrement d'emprunts, le trigger met à jour le champ Nbrexemplaires disponibles pour les livres concernés
- **TR 6.** A l'enregistrement de retours de livres, Ce trigger met à jour le champ Nbrexemplaires disponibles pour les livres concernés
- **TR 7.** A la suppression d'emprunts, ce trigger doit vérifier si aucun retour correspondant à ces emprunts n'a été enregistré. Si c'est le cas, la suppression est validée et le champ Nbrexemplaires disponibles pour les livres concernés doit être mis à jour
- **TR 8.** A la modification d'emprunts, ce trigger doit mettre à jour le champ Nbrexemplaires disponibles pour les livres concernés

Exercice 5

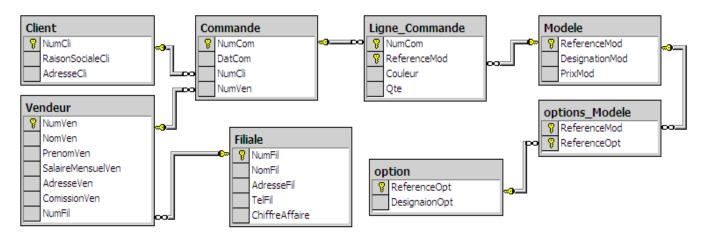
La société SAMIRAUTO est spécialisée dans la vente de véhicules automobiles. Cette société a un ensemble de filiales réparties sur le royaume :

- A chaque filiale est rattaché un ensemble de vendeurs
- Les vendeurs doivent convaincre des clients de passer des commandes pour l'achat de modèles de véhicules
- Un modèle de véhicule est disponible avec un ensemble d'options

Naoual ABDALLAH	Page : 39	Version Octobre 2007

- Un même modèle peut être disponible en plusieurs couleurs
- Dans une commande, on indique pour chaque modèle, la quantité commandée et la couleur souhaitée

Soit la base de données suivante :



Créer les procédures stockées suivantes :

- PS 1. Qui affiche le modèle de véhicule le moins cher
- **PS 2.** Qui reçoit la référence d'un modèle et affiche la liste des options (Référence et Désignation) qui lui sont associées
- **PS 3.** Qui reçoit un numéro de vendeur et retourne le nombre de commandes qu'il a effectué
- **PS 4.** Qui reçoit un numéro de vendeur. Si ce vendeur n'a décroché aucune commande, afficher le message 'Mauvaise performance', si ce vendeur a décroché plus de 20 commandes, afficher le message 'Haute performance' sinon afficher le message 'Performance normale' (utiliser PS_3)
- **PS 5.** Qui reçoit une référence modèle et qui affiche par couleur le nombre total de modèles commandés
- **PS 6.** Qui reçoit un numéro de vendeur et un numéro de mois (chiffre de 1 à 12) et qui retourne son salaire (le salaire d'un vendeur se compose d'un salaire mensuel fixe auquel on ajoute un pourcentage (CommissionVen) sur le montant total des commandes que ce vendeur a réalisé au cours du mois donné en paramètre)
- **PS 7.** Qui pour chaque vendeur affiche le numéro, le nom et la liste des commandes effectuées (numéro et date)
- **PS 8.** Qui reçoit un numéro de commande. Si cette commande n'existe pas, affiche un message d'erreur 'Cette commande n'existe pas'. Si cette commande existe affiche pour cette commande :
 - ✓ Le numéro et la date de commande
 - ✓ La liste des Modèles Commandés avec pour chaque modèle la référence, la désignation et la liste des options associées
- **PS 9.** Qui reçoit un numéro de vendeur et un numéro de client et qui affiche (utiliser PS_8):
 - ✓ Les informations sur le vendeur (nom et adresse)
 - ✓ Les informations sur le client (Raison sociale et adresse)

Naoual ABDALLAH	Page : 40	Version Octobre 2007

- ✓ La liste des commandes avec pour chaque commande :
 - o Le numéro et la date de commande
 - La liste des Modèle Commandés avec pour chaque modèle la référence, la désignation et la liste des options associées

Créer les triggers suivants :

- **TR 1.** Qui empêche la création de filiales ayant le même nom, la même adresse ou le même numéro de téléphone
- TR 2. Qui interdit la modification des informations sur une filiale
- **TR 3.** Qui interdit la modification du numéro de commande dans une ligne de commande
- **TR 4.** Dans la table LigneCommande, le champ couleur ne peut prendre que Gris, noir ou bleu et la quantité doit être strictement supérieure à 0
- **TR 5.** A l'enregistrement d'une ligne de commande, mettre à jour le champ ChiffreAffaire
- **TR 6.** A la suppression d'une ligne de commande, mettre à jour le champ ChiffreAffaire
- **TR7.** A la modification d'une ligne de commande, mettre à jour le champ ChiffreAffaire

LISTE DES REFERENCES

Documentation Microsoft:

- Aide Transact-SQL à partir de SQL Server
- Aide SQL Server 2000

Naoual ABDALLAH	Page : 41	Version Octobre 2007

ANNEXE 1: SOLUTION SERIE D'EXERCICES / PROCEDURES STOCKEES

Exercice 1

CREATE PROCEDURE PS1 AS

Select NumCostume, DesignationCostume, NomStyliste, AdrStyliste from Styliste, Costume where Styliste.NumStyliste=Costume.NumStyliste

CREATE PROCEDURE PS2 @NumCos int AS

Select DesignationCostume, NomStyliste, AdrStyliste from Styliste, Costume where Styliste.NumStyliste = Costume.NumStyliste and NumCostume=@NumCos

CREATE PROCEDURE PS3 @NumCos int AS

Select MembreJury.NumMembreJury, NomMembreJury, FonctionMembreJury, NoteAttribuée from MembreJury, Notesjury where MembreJury.numMembreJury= Notesjury.NumMembreJury and NumCostume =@NumCos

CREATE PROCEDURE PS4 @NbrCos int output AS

Set @NbrCos=(select count(NumCostume) from Costume)

CREATE PROCEDURE PS5 @NumCos int, @MJ int, @note decimal output AS

Set @note=(select Noteattribuée from Notesjury where NumCostume =@NumCos and NumMembreJury=@MJ)

CREATE PROCEDURE PS6 @NumCos int, @M decimal output AS

Set @M=(select Avg(NoteAttribuée) from Notesjury where NumCostume =@NumCos)

Exercice 2

CREATE PROCEDURE SP1 AS

Create table ProduitBrut(CodProBrut int primary key,NomProBrut varchar(50), PrixAchat decimal, NumFour int Foreign Key references Fournisseur)

Create table Composition (CodProFini int Foreign key references ProduitFini, CodProBrut int Foreign Key references ProduitBrut, QteUtilisee decimal, Constraint PK_Composition Primary Key(CodProFini, CodProBrut))

CREATE PROCEDURE SP2 AS

Select codProFini, count(CodProBrut) from composition Group by CodProFini

CREATE PROCEDURE SP3 @MaxPrix decimal output AS

Set @MaxPrix = (Select max(PrixAchat) from ProduitBrut)

CREATE PROCEDURE SP4 AS

Select CodProFini from Composition Group by CodProFini

Having Count(CodProBrut)>=2

CREATE PROCEDURE SP5 @ProBrut varchar(50), @RS varchar(50) output AS

Set @RS=(Select RSFour From Fournisseur, ProduitBrut

Where Fournisseur.NumFour= ProduitBrut.NumFour and NomProBrut=@ProBrut)

CREATE PROCEDURE SP6 @CodProFini int AS

Select * from Mouvement Where TypeMvt='S' and CodProFini=@CodProFini

CREATE PROCEDURE SP7 @CodProFini int, @TypeMvt char(1) AS

Select * from Mouvement Where TypeMvt=@TypeMvt and CodProFini=@CodProFini

CREATE PROCEDURE SP8 AS

Declare @CPF int, @QteStock decimal

Declare C1 Cursor for select CodProFini, QteEnStock from ProduitFini

Naoual ABDALLAH	Page : 42	Version Octobre 2007

```
open C1
 Fetch next from C1 into @CPF, @QteStock
 while @@fetch_status=0
       Begin
             Print 'La quantité en stock est : ' + convert(varchar, @QteStock)
             Select * from Mouvement where CodProFini=@CPF
             Declare @SommeE decimal, @SommeS decimal
                  @SommeE=(Select Sum(Quantite)
                                                      from
                                                             Mouvement
                                                                          where
                           TypeMvt="E" and CodProFini=@CPF)
                   @SommeS=(Select
             Set
                                      Sum(Quantite)
                                                      from
                                                             Mouvement
                                                                          where
                           TypeMvt="S" and CodProFini=@CPF)
             if @SommeE-@SommeE <>@QteStock
                   Print 'Stock OK'
             Else
                   Print 'Problème de Stock'
       Fetch Next from C1 into @CPF, @QteStock
 Close C1
 Deallocate C1
CREATE PROCEDURE SP9 @CPF int, @PrixReviens decimal output AS
      @PrixReviens=(select
                            Sum(PrixAchat*Qteutilisee)
                                                                            PB,
                                                       from
                                                               ProduitBrut
 Composition C where C.CodProBrut=PB.CodProBrut and CodProFini=@CPF)
CREATE PROCEDURE SP10 AS
 Declare @CPF int, @NbrProduitsFinis int
 Declare C1 Cursor for select CodProFini, count(CodproBrut) from Composition
 group by CodProFini
 Open C1
 Fetch Next from C1 into @CPF, @NbrProduitsFinis
 While @@fetch_status=0
    Begin
       Declare @PR decimal
       Exec SP9 @CPF, @PR output
       Print 'Le prix de reviens est : ' + convert(varchar, @PR)
       Select NomProFini, PrixAchat*Qteutilisee, RSFour From ProduitBrut PB,
       Composition C Where PB.CodProBrut=C.CodProBrut and CodProFini=@CPF
       Print 'Le nombre de produits bruts est : ' + convert(varchar,@NbrProduitsFinis)
       Fetch Next from C1 into @CPF, @NbrProduitsFinis
    End
 Close C1
 Deallocate C1
 Exercice 3
CREATE PROCEDURE SP1 AS
```

Select NumStagiaire, NomStagiaire From Stagiaire where NumStagiaire not in (Select NumStagiaire from Notation,)

CREATE PROCEDURE SP2 AS

Select Filiere. NumFiliere, NomFiliere from filiere, programme

Naoual ABDALLAH	Page : 43	Version Octobre 2007

Where Filiere.numFiliere=Programme.Numfiliere Group by Filiere. NumFiliere, NomFiliere Having Count(NumModule)>=10 CREATE PROCEDURE SP3 @CodSecteur varchar(10) AS Select NomModule from Module M,Programme P, Filiere F M.Nummodule=P.numModule and P.NumFiliere=F.NumFiliere Where and codSecteur=@CodSecteur group by NomModule Having count(F.numfiliere)=(select Count(numfiliere) from filiere where CodSecteur=@CodSecteur) CREATE PROCEDURE SP4 @NumStagiaire int AS Select Module, NomModule, Note, Coefficient from Module, Notation, programme Where Notation.NumModule=Module.NumModule and Module.Nummodule=programme.Nummodule and numStagiaire=@NumStagiaire **CREATE PROCEDURE SP5 AS** Declare @NumSta int, @NomPreSta varchar(50), @NomFil varchar(50) Declare C1 Cursor for Select NumStagiaire, NomStagiaire + ' ' + PrenomStagiaire, NomFiliere from Stagiaire S, Filiere F where S.NumFiliere=F.NumFiliere Open C1 Fetch Next from C1 into @NumSta, @NomPreSta, @NomFil while @@fetch_status=0 Begin Print 'Nom et Prénom : ' + @NomPreSta + ' Filière : ' + @NomFil if exists (select NumModule from Programme P, Stagiaire S where P.NumFiliere=S.NumFiliere and NumStagiaire=@NumSta NumModule not in (select NumModule from notation where NumStagiaire=@NumSta)) Begin Print 'En cours de traitement' select NumModule from Programme P, Stagiaire S where P.NumFiliere=S.NumFiliere and NumStagiaire=@NumSta and NumModule not in (select NumModule from notation where NumStagiaire=@NumSta) Fetch Next from C1 into @NumSta, @NomPreSta, @NomFil Continue if (select count(NumModule) from notation where NumStagiaire = @NumSta and note<3) >2Begin Print 'Notes Eliminatoires' Select NomModule from Module M, Notation N where

Naoual ABDALLAH	Page: 44	Version Octobre 2007
Navuai Addallai i	raye. ++	Version Octobre 2007

M.NumModule=N.NumModule

numStagiaire=@NumSta and note <3

Fetch Next from C1 into @NumSta, @NomPreSta, @NomFil
Continue

End

Select NomModule, Coefficient, Note from Module M, Programme P, where M.NumModule=P.NumModule and Notation N, Stagiaire S P.numFiliere=S.NumFiliere and M.NumModule=N.NumModule N.NumStagiaire=S.NumStagiaire and S.numStagiaire=@NumSta Select Sum(Note*coefficient) / Sum(coefficient) from Module M, Programme Ρ, Notation N, Stagiaire S where M.NumModule=P.NumModule and P.numFiliere=S.NumFiliere and M.NumModule=N.NumModule and N.NumStagiaire=S.NumStagiaire and S.numStagiaire=@NumSta

Fetch Next from C1 into @NumSta, @NomPreSta, @NomFil End

Close C1

Deallocate C1

ANNEXE 2: SOLUTION SERIE D'EXERCICES / TRIGGERS

Exercice 1

Rappel : Les triggers After ne se déclenchent pas si les contraintes d'intégrité ne sont pas validées

CREATE TRIGGER TRI ON COSTUME INSTEAD OF INSERT AS

if not exists(select NumStyliste from inserted where NumStyliste not in(select NumStyliste from styliste))

insert into Costume select * from inserted

CREATE TRIGGER TRI ON COSTUME FOR INSERT AS

if exists(select NumStyliste from inserted where NumStyliste not in(select NumStyliste from styliste))

Rollback

CREATE TRIGGER TR2 ON COSTUME FOR DELETE AS

if exists(select NumCostume from deleted where NumCostume in (select NumCostume from notejury))

Rollback

CREATE TRIGGER TR2 ON COSTUME INSTEAD OF DELETE AS

if not exists(select NumCostume from deleted where NumCostume in (select NumCostume from notejury))

Delete from Costume where NumCostume in (select NumCostume from Deleted)

CREATE TRIGGER TR3 ON NOTEJURY FOR INSERT AS

If exists(select NumCostume from inserted where NumCostume not in (select NumCostume from costume) or exists (select NumMemberJury from inserted where NumMemberJury not in (select NumMembreJury from MembreJury) or exists(select note from inserted where note not between 0 and 20) Rollback

CREATE TRIGGER TR3 ON NOTEJURY INSTEAD OF INSERT AS

If not exists(select NumCostume from inserted where NumCostume not in (select NumCostume from costume) and not exists (select NumMemberJury from inserted where NumMemberJury not in (select NumMembreJury from MembreJury) and not exists(select note from inserted where note not between 0 and 20) Insert into NoteJury select * from inserted

CREATE TRIGGER TR4 ON MEMBREJURY FOR INSERT AS

insert into fonction select FonctionMembre from inserted where fonctionMembre not in (select fonction from fonction)

Exercice 2

CREATE TRIGGER TR1 ON CLIENT FOR INSERT, UPDATE AS

if exists (select CINCli from client group by CINCli Having count(numcli)>1) Rollback

Naoual ABDALLAH	Page : 46	Version Octobre 2007

CREATE TRIGGER TR1 ON CLIENT INSTEAD OF INSERT, UPDATE AS

if not exists (select CINCli from inserted where CINCli not in (Select CINCli from Client))

Begin

If (select Count(NumCli) from Deleted) =0)

Insert into client select * from inserted

Else

Update Client set Client.CINCli=I.CINCli, Client.NomCli=I.NomCli, Client.AdrCli=I.AdrCli, Client.TelCli=I.TelCli where Client.NumCli=I.NumCli and I.NumCli=D.NumCli From Inserted I Where Client.NumCli=I.NumCli End

CREATE TRIGGER TR2 ON COMPTE FOR INSERT AS

if exists(select NumCpt from inserted where SoldeCpt <1500) or exists (select NumCpt from inserted where Typecpt <>'CC' and TypeCpt <> 'CN') or exists(select inserted.numcli from inserted,compte where inserted.numcli= compte.numcli and compte.numcpt <> inserted.numcpt and compte.typecpt=inserted.typecpt)

Rollback

Ou Autre solution

if exists(select NumCpt from inserted where SoldeCpt <1500) or exists (select NumCpt from inserted where Typecpt <>'CC' and TypeCpt <> 'CN') or exists(select Numcli, TypeCpt from compte group by NumCli, TypeCpt Having Count(NumCpt)>1)

Rollback

CREATE TRIGGER TR2 ON COMPTE INSTEAD OF INSERT AS

if not exists(select NumCpt from inserted where SoldeCpt <1500) and not exists (select NumCpt from inserted where Typecpt <>'CC' and TypeCpt <> 'CN') and not exists(select inserted.numcli from inserted,compte where inserted.numcli= compte.numcli and compte.typecpt=inserted.typecpt)

insert into compte select * from inserted

CREATE TRIGGER TR3 ON COMPTE FOR DELETE AS

if exists (select SoldeCpt from deleted where soldeCpt>0)

Rollback

if exists(select numcpt from operation group by numcpt Having Datediff(month, max(dateOP), getdate())<3)

Rollback

CREATE TRIGGER TR4 ON COMPTE FOR UPDATE AS

if update(numcpt)

Rollback

if exists(select inserted.solde from inserted, deleted, operation where inserted.numcpt=deleted.numcpt and deleted.numcpt=operation.numcpt and deleted.solde<>inserted.solde)

Rollback

if exists(select inserted.numcpt from inserted, deleted where inserted.numcpt=deleted.numcpt and deleted.typecpt='cn' and inserted.typecpt='cc')

Naoual ABDALLAH	Page : 47	Version Octobre 2007

```
Rollback
```

if exists(select inserted.numcli from inserted,compte where

inserted.numcli=compte.numcli and compte.numcpt<>inserted.numcpt and
compte.typecpt=inserted.typecpt)

Begin

Rollback

End

Exercice 3

CREATE TRIGGER TRI ON PRODUITBRUT FOR INSERT AS

Update fournisseur set NbrProduitsfournis=NbrProduitsfournis+(select count(Codprobrut) from inserted where

inserted.numfour fournisseur.numfour)

from inserted, fournisseur where inserted.numfour=fournisseur.numfour

CREATE TRIGGER TR2 ON PRODUITBRUT FOR DELETE AS

Update fournisseur set NbrProduitsfournis = NbrProduitsfournis -(select count(Codprobrut) from deleted where deleted.numfour=fournisseur.numfour)

From deleted, fournisseur

Where deleted.numfour=fournisseur.numfour

CREATE TRIGGER TR3 ON MOUVEMENT FOR INSERT AS

Update Produitfini set QteEnstock=QteEnstock+ (select Sum(Quantite) from inserted where inserted.codprofini= produitfini.codprofini and Typemvt='e')

from inserted, Produitfini

Where inserted.codprofini=produitfini.codprofini and Typemvt='e'

Update Produitfini set QteEnstock = QteEnstock -(select sum(Quantite) from inserted

where inserted.codprofini=

produitfini.codprofini and Typemvt='s')

From inserted, Produitfini

Where inserted.codprofini=produitfini.codprofini and Typemvt='s

CREATE TRIGGER TR4 ON MOUVEMENT FOR DELETE AS

Update Produitfini set QteEnstock = QteEnstock -(select sum(Quantite) from deleted where deleted.codprofini= produitfini.codprofini and Typemvt='e')

From deleted, Produitfini

Where deleted.codprofini=produitfini.codprofini and Typemvt='e'

Update Produitfini set QteEnstock = QteEnstock +(select sum(Quantite) from deleted where deleted.codprofini= produitfini.codprofini and Typemvt='s')

From deleted, Produitfini

Where deleted.codprofini=produitfini.codprofini and Typemvt='s'

Naoual ABDALLAH	Page : 48	Version Octobre 2007

CREATE TRIGGER TR5 ON MOUVEMENT FOR UPDATE AS Première solution (Trop longue)

Update Produitfini set QteEnstock = QteEnstock +(select sum(Quantite) from deleted where deleted.codprofini=produitfini.codprofini and deleted.typeop='s')-(select sum(Quantite) from inserted where inserted.codprofini= produitfini.codprofini and inserted.typeop='s')

From inserted , Produitfini, deleted

Where inserted.codprofini=produitfini.codprofini and deleted.codprofini=produitfini.codprofini and inserted.typeop='s' and deleted.typeop='s'

update Produitfini set QteEnstock = QteEnstock -(select sum(Quantite) from deleted where deleted.codprofini=produitfini.codprofini and deleted.typeop='e')+(select sum(Quantite) from inserted where inserted.codprofini= produitfini.codprofini and inserted.typeop='e')

From inserted ,Produitfini,deleted

Where inserted.codprofini=produitfini.codprofini and deleted.codprofini=produitfini.codprofini and inserted.typeop='e' and deleted.typeop='e'

Update Produitfini set QteEnstock = QteEnstock +(select sum(Quantite)from deleted where deleted.codprofini=produitfini.codprofini and deleted.typeop='e')+(select sum(Quantite)from inserted where inserted.codprofini<> produitfini.codprofini and inserted.typeop='s')

From inserted ,produitfini,deleted

Where inserted.codprofini=produitfini.codprofini and deleted.numcmp=produitfini.numcpt and inserted.typeop='s' and deleted.typeop='e'

Update Produitfini set QteEnstock = QteEnstock -(select sum(Quantite)from deleted where deleted.codprofini=produitfini.codprofini and deleted.typeop='s')-(select sum(Quantite)from inserted where inserted.codprofini<> produitfini.codprofini and inserted.typeop='e')

From inserted ,Produitfini,deleted

Where inserted.codprofini=produitfini.codprofini and deleted.numcmp=produitfini.numcpt and inserted.typeop='e' and deleted.typeop='s'

CREATE TRIGGER TR5 ON MOUVEMENT FOR UPDATE AS Deuxième solution

update produitfini set QteEnstock=QteEnStock - (select sum(Quantite) from deleted Where deleted.CodProFini=ProduitFini.CodProFini and TypeMvt='E')

from produitfini, deleted where produitfini.codprofini=deleted.codprofini and typeMvt='E'

Naoual ABDALLAH	Page : 49	Version Octobre 2007

update produitfini set QteEnstock=QteEnStock + (select sum(Quantite) from deleted Where deleted.CodProFini=ProduitFini.CodProFini and TypeMvt='S')

from produitfini, deleted where produitfini.codprofini=deleted.codprofini and typeMvt='S'

update produitfini set QteEnstock=QteEnStock - (select sum(Quantite) from inserted Where

Inserted.CodProFini=ProduitFini.CodProFini and TypeMvt='S')

from produitfini, inserted where produitfini.codprofini=inserted.codprofini and typeMvt='S'

update produitfini set QteEnstock=QteEnStock + (select sum(Quantite) from inserted Where

Inserted.CodProFini=ProduitFini.CodProFini and TypeMvt='E')

from produitfini, inserted where produitfini.codprofini=inserted.codprofini and typeMvt='E'

Exercice 4

Les triggers TR5, TR6 et TR7 de cet exercice seront résolus respectivement de la même façon que les triggers TR3, TR4 et TR5 de l'exercice précédent en opérant les modifications suivantes :

- Remplacer la table ProduitFini par Compte
- Le champ CodProFini par NumCpt
- Le Champ TypeMvt par TypeOp
- Le Champ Quantite par MtOp
- Le Champ QteEnStock par SoldeCpt
- Le caractère 'S' (sortie) par 'R' (Retrait) et le caractère 'E' (Entrée) par 'D' (Dépôt)

Naoual ABDALLAH	Page : 50	Version Octobre 2007