Transact SQL

Pr H.LAARAJ

Soit le schéma de la base de données « GCommande » suivante :

1. Notions en Transact Sql
   a. Déclaration des variables

      Syntaxe :

      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 @.

   b. L'affectation d'une valeur à une variable

      Syntaxe:

      Select @Nom_Variable = valeur

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

      ou

      Set @Nom_Variable = valeur

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

      Exemples :

```
Set @a=1
-- Affecte la valeur 1 à la variable @a
Set @a = (Select count (NumArt) from Article)
 -- Affecte le nombre d'articles enregistrés dans la table article à la variable @a
Set @b='commande'
-- Affecte la valeur 'commande' à la variable @b
```

c. 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 @c + @d
  -- Affiche la somme des variables @c et @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)

d. Les 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 int
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'
```

e. L'instruction case

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 :

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
```

Exercice 1 :

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.

2. Les 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
```

3. 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'
```

4. La transaction

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
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
```

5. 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.

6. les curseurs

a. définition

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 :

- **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, qu'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**

  Declare nom_curseur **Cursor** | Static
Keyset
Dynamic | **For Select …**

- **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

  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 n'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 désignation ……coûte …. …..

```
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
```

Exercice :

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

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