معهد البرمجة و الشبكات

Table des matières

Les variables

Une variable est une zone mémoire caractérisée par un type et un nom, et permettant de stocker une valeur respectant le type. Dans SQL Server, les variables doivent être obligatoirement déclarées avant d'être utilisée.

Voici la déclaration d'une variable nommée Id_Client de type Int :

```
DECLARE @IdClient int
```

L'instruction suivante permet de valoriser cette variable via l'exécution d'une requête scalaire :

```
SELECT @IdClient = (SELECT Id_Client FROM Motors.dbo.Client WHERE
```

```
declare @a int= (select max(Age) from stg)
print 'L'Age maximum est :'+convert(varchar(20),@a)

DECLARE @date_debut datetime <== Déclaration
DECLARE @date_fin datetime
SET @date_debut='1/1/1997' <==  Affectation
SET @date_fin='1/20/1997'
```

La clause if

```
If (@a>10)
Begin  print 'Age grand'  End
Else
Begin  print 'Age petit'  End
```

La clause while

```
While Condition
Begin
instructions
End
```

La clause case

```
select NomClient,
    case idClient
    when 1 then 'num 1'
    when 2 then 'num 2'
    when 3 then 'num 3'
    when 4 then 'num 4'
    end
from client
```

La clause tran

```
Begin Tran
…
If Condition
RollBack
…
Commit
```

Sélectionne un champs selon condition

```
select *,'type prix'=
case
when PUArt <=200 then 'Pas chère'
when PUArt >200 and PUArt <900 then
'Moyen'
when PUArt >=900 then 'chère'
end
from Article
```

Ajouter un champs Calculer

```
alter table article add type_prix as
case
when PUArt <=200 then 'Pas chère'
when PUArt >200 and PUArt <900 then
'Moyen'
when PUArt >=900 then 'chère'
end
```

Les procédures stockées

Introduction aux procédures stockées

Les procédures stockées sont des ensembles d'instructions du DML, pouvant être exécutés par simple appel de leur nom ou par l'instruction **EXECUTE** .

Les procédures stockées sont de véritables programmes qui peuvent recevoir des paramètres, être exécutés à distance, renvoyer des valeurs et possédant leurs propres droits d'accès **(EXECUTE).** Celles-ci sont compilées une première fois, puis placées en cache mémoire, ce qui rend leur exécution plus performante du fait que le code soit précompilé. Les procédures stockées sont contenues dans la base de données, et sont appelable par leurs noms.

Création d'une procédure stockée

```
CREATE PROCEDURE Ajout_Client
@Nom varchar(50), @Prenom varchar(50),
@Numero varchar(50), @Adresse varchar(50),
@Mail varchar(50)
AS
BEGIN
      INSERT INTO Client
      (Nom_Client,
      Prenom_Client,
      Numero_Client,
      Adresse_Client,
      Mail_Client)
      VALUES
      (@Nom,
      @Prenom,
      @Numero,
      @Adresse,
      @Mail)
END
GO
```

Il est alors possible d'utiliser cette procédure stockée de la façon suivante :

```
EXEC dbo.Ajout_Client 'NARBONNE', 'Christophe',
33678764534, '17 allée des embrumes', NULL
```

Remarque : Il est obligatoire de mettre les arguments de la procédure stockée dans l'ordre dont elles sont décrites dans la procédure stockée. Si toute fois vous ne voulez pas les mettre dans l'ordre, il est possible de préciser le nom de la variable et d'y assigner une valeur de la façon suivante :

```
EXEC dbo.Ajout_Client @Nom = 'NARBONNE', @Prenom = 'Christophe',
@Numero = 33678764534, @Mail = NULL, @Adresse = '17 allée des embrumes'
```

Suppression d'une procédure stockée

```
USE Entreprise
DROP PROCEDURE Ajout_Client
```

Procédure stocké avec paramètre en output

```
alter proc sp_nbr_cmd @Nbr int output
as
Set @Nbr= (select count(*) from commande )

declare @a int
exec sp_nbr_cmd @a output
print 'Le nombre totale des commande est :' + convert(varchar,@a )
```

Les Fonction

Les fonctions utilisateurs (UDF, autrement dit **User Define Function)** sont de plusieurs types. Trois pour être précis. Il y a les fonctions scalaires, les fonctions tables en ligne et les fonctions multi-instructions. Une fonction peu accepter des arguments, et ne peu retourner que deux types de données : une valeur scalaire ou une table.

- Les fonctions scalaires retournent, grâce au mot clé RETURN, une valeur scalaire. Tous les types de données peuvent être retournés par une fonction scalaire hors mis timestamp, table, cursor, text, ntext et image.
- Les fonctions table ne retourne comme résultat, qu'une table, qui est le résultat d'une instruction SELECT.

LES FONCTION RENVOYANT UNE VALEUR : SCALAIRES

exemple d'une fonction qui retourne l'Age

```
Create function date_age(@date date)
        Returns int
        As
        Begin
        Return DateDiff (Year, @date, GetDate())
        End
```

Appel d'une fonction :

```
Select dbo.date_age ('19900913')
```

LES FONCTION RENVOYANT UNE TABLE

- Fonction table simple :

```
CREATE FUNCTION recommander_stock (@Id int, @seuil int)
RETURNS TABLE
AS
RETURN (SELECT * FROM Stock WHERE Id_Stock = @Id AND Quantite < @Seuil)
```

- Fonction table multi-instruction :

```
CREATE FUNCTION table_multi (@Id int)
RETURNS @variable TABLE (Id_Stock int, Quantite int, Nom_Entrepos
varchar(25))
AS
BEGIN
SELECT @variable = (SELECT Id_Stock, Quantite, Nom_Entrepos
FROM Entrepos E INNER JOIN Stock S
ON E.Id_Entrepos = S.Id_Entrepos
WHERE S.Id_Stock = @Id)
RETURN
END
```

- Modification d'une fonction

```
ALTER FUNCTION nombre_element_stock (@Entrepos int)
RETURNS int
AS
BEGIN
DECLARE @nb int
SELECT @nb = COUNT(Id_Stock)
FROM Stock
WHERE Id_Entrepos = @Entrepos
RETURN @nb
END
GO
```

- Suppression d'une fonction

```
DROP FUNCTION nombre_element_stock
```

Fonction retourne une table simple :

```
create function fun2 (@qtt int)
returns table
as
return (select * from article where
qteenstock<10)
```

Cette fonction retoune une table :article avec la condition where le qtt en stock inferieur a un parametre

```
select * from dbo.fun2(100)
```

Fonction de type table compose :

```
create function select_artiicle(@id
int)
returns @tab table(description
varchar(20),Prix money,qttStock int)
as
begin

insert into @tab select
desart,puart,qteenstock from article
where numart=@id

return
end
```

```
select * from dbo.select_artiicle(3)
```

Les triggers

Un trigger (déclencheurs en français) est un programme qui se déclenche automatiquement suite à un évènement

A la différence d'une procédure stockée, on ne peut pas appeler un trigger explicitement.

En base de données, l'évènement est une instruction du DML qui modifie la base. (INSERT, DELETE, UPDATE)

Ces triggers font partie du schéma de la base. Leur code compilé est Conservé (comme pour les programmes stockes)

- **Pourquoi utiliser les triggers**

- Les triggers peuvent servir à vérifier des contraintes que l'on ne peut pas définir de façon déclarative
- Ils peuvent aussi gérer de la redondance d'information.
- Ils peuvent aussi servir à collecter des informations sur les mises à jour de la **BD**

- Il peut y avoir des triggers de 3 types :INSERT/DELETE/UPDATE

Nb :

- on cas **d'insertion** d'un enregistrement le trigger d'insertion crée une table appeler **inserted** (table virtuelle) cette table contient l'enregistrement à insérer quand le trouve aussi dans la table réel
- on cas de **suppression** le moteur de base de donnée crée la table **deleted** cette table contient l'enregistrement à supprimer et la table d'origine réel ne contient plus cette enregistrement
- on cas de **modification** la table **inserted** et la table d'origine contient la modification, la table **deleted** contient l'ancien enregistrement

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 ;
- Un seul déclencheur **INSTEAD OF** est autorisé par action de déclenchement dans une table

Les déclencheurs **AFTER** :

- Sont exécutés après la validation des contraintes associées à la table.
- Ne peuvent être associés qu'à des tables
- Plusieurs déclencheurs AFTER sont autorisés sur une même table et pour une même action de déclenchement.

Trigger qui interdit de modifier un champs
Nb :
`if(update(Num))c.-à-d. testé si la valeur de num est changer`
	`alter trigger essai_trigger on Article`
	`after update as`
	`begin`
	`  if(update(puart))`
	`  begin`
	`  print 'Impossible de modifier le Prix'`
	`  rollback`
	`  end`
	`end`

trigger suivant à l'ajout d'une ligne de commande vérifie si les quantités sont disponibles et met le stock à jour	`Create trigger verif_qtt on LigneCommande`
	`after insert as`
	`begin`
	`declare @nart int=(select numart from inserted)`
	`declare @stock int=(select qteenstock from Article where numart=@nart)`
	`declare @cmd int=(select qtecommandee from inserted)`
	`if(@stock<@cmd)`
	`begin`
	`print'la quantité est insufisant'`

	rollback
	end
	else
	begin
	update Article set qteenstock=qteenstock-@cmd
	where numart=@nart
	print'qtt stock modifier avec succes'
	end
	end

Créer trigger	Supprimer un trigger
Create Trigger insert_client On client after insert As Begin Declare @nom varchar(50) Set @nom=(select nom from inserted) Print 'le Client :'+@nom+'est ajouter ' End	drop trigger insert_client

Exercice

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 proc SP_EnregistrerLigneCom
	@NumCmd int,@NumArt int,@QteCmd int
	as
	begin
	if(not exists(select numart from
	Article where numart=@NumArt))
	print 'L article n existe pas'
	else
	if(@QteCmd>(select qteenStock from
	article where numart=@NumArt))
	print 'la quantité demandée n est
	pas disponible'
	else
	begin tran
	insert into commande values
	(@NumCmd,getdate())
	insert into LigneCommande
	values(@NumCmd,@NumArt,@QteCmd)
	update article set qteenstock
	=qteenstock-@QteCmd where numart
	=@NumArt
	commit
	end
	
	exec SP_EnregistrerLigneCom 111,5,10

Les curseur :

```
declare cur_1 scroll cursor for select nomclient,prenomclient,adrclient
from client--declarer Curseur--
open cur_1--ouverture de Curseur--
declare @Nom varchar(20),@Prenom Varchar(20),@Adresse varchar(20)--
declaration des variables--
fetch first from cur_1 into @Nom,@Prenom,@Adresse--deplacement du curseur--
while(@@FETCH_STATUS = 0)--la boucle avec le variable @@feetch_status
renvois 0 s'il trouve un enregistrement -1 s'il ne trouve rien--
begin
print 'Le Nom : '+@Nom+' Le Prenom : '+@Prenom+' L adresse  : '+@Adresse
fetch next from cur_1  into @Nom,@Prenom,@Adresse
end

close cur_1 --fermeture de Curseur--
deallocate cur_1 -- suppresion de Curseur--

--defilement:
--forward-only: deplacement vers l'avant uniquement
--fast-forward: definir le curseur en avant et en lecture
--scroll: deplacement dans tous les sens
--Read-only: En lecture seul

--positionnement d'un curseur:
--first: Le Pemier
--last: Le Dernier
--next: la ligne aprés la ligne courante
--prior: la ligne avant la ligne courante


declare cur1 cursor scroll for select NumArt,DesArt,PuArt,QteEnStock from
article
open cur1
declare @num int,@desc varchar(20),@prix money,@Quantite int
fetch first from cur1 into @num,@desc,@prix,@Quantite

while(@@FETCH_STATUS=0)
begin
print 'Le Num  :'+convert(varchar(20),@num)+' Description  :'+@desc+' Prix
:'+convert(varchar(20),@prix)
fetch next from  cur1 into @num,@desc,@prix,@Quantite

end

close cur1
```