



M21 : SGBD 2

Transact-SQL(P.S, Fonction & Triggers)

Formateur : DRIOUCH B.

Etablissement : OFPPT/GC/CFMOTI (15/11/2012)
cfmoti.driouch@gmail.com
<http://www.ista-ntic.net/>

Plan

- Introduction - Définition
- Types de données
- Instruction (Déclaration, Affectation, Affichage)
- Instruction de contrôle (Alternative, Itérative)
- Gestion transaction – Erreur (Exception) →
- Les Curseurs →
- Les Procédures Stockés →
- Les Fonctions →
- Les Triggers →

Transact-SQL

■ Définition

Transact-SQL est une extension de SQL vers un langage de programmation, il est aussi le point central de l'utilisation de Microsoft SQL Server. Toutes les applications qui communiquent avec SQL Server le font en envoyant des instructions Transact-SQL au serveur, quelle que soit l'interface utilisateur de l'application.

En plus de ça, pour soulager les postes client, une partie des applications client on les programme au niveau Serveur (SGBD) avec plus de performance processeur et mémoire.

Types de Données

Valeurs numériques exactes		Stockage
bigint	De -2^{63} (-9 223 372 036 854 775 808) à $2^{63}-1$ (9 223 372 036 854 775 807)	Huit octets
int	De -2^{31} (-2 147 483 648) à $2^{31}-1$ (2 147 483 647)	Quatre octets
smallint	De -2^{15} (-32 768) à $2^{15}-1$ (32 767)	Deux octets
tinyint	De 0 à 255	Un octet
bit	Données de type entier qui peuvent prendre la valeur 1, 0 ou NULL. Les valeurs de chaînes TRUE et FALSE peuvent être converties en bit : TRUE est converti en 1 et FALSE en 0	1 ou 2 octets
Decimal, numeric	Types de données numériques ayant une précision et une échelle fixes. decimal[(p[, s])] et numeric[(p[, s])]	
money	-922 337 203 685 477,5808 à 922 337 203 685 477,5807	8 octets
smallmoney	-214 748,3648 à 214 748,3647	4 octets

Types de Données

Valeurs numériques approximatives		
float [(n)]	- 1,79E+308 à -2,23E-308, 0 et 2,23E-308 à 1,79E+308	Selon la valeur de n
real	- 3,40E + 38 à -1,18E - 38, 0 et 1,18E - 38 à 3,40E + 38	4 octets

Date et heure		
Datetime	Du 1er janvier 1753 au 31 décembre 9999	8 octets
smalldatetime	Du 1er janvier 1900 au 6 juin 2079	4 octets

Chaînes de caractères		
[(n)]	d'une longueur fixe de n octets, n doit être compris entre 1 et 8000	Entre 1 et 8000 octets
varchar[(n max)]	Données de type caractère non-Unicode d'une longueur variable. n doit être compris entre 1 et 8000. max indique que la taille maximale de stockage est égale à $2^{31}-1$ octets	Selon n
text	Données non-Unicode de longueur variable figurant dans la page de codes du serveur et ne pouvant pas dépasser en longueur $2^{31} - 1$ (2 147 483 647) caractères	Le stockage est tout de même de 2 147 483 647 octets

DRIOUCH B.

5

Types de Données

Chaînes de caractères Unicode		
nchar	Données de type caractères Unicode de longueur fixe de n caractères. n doit être compris entre 1 et 4 000	La taille de stockage, en octets, est le double de n.
Nvarchar[(n max)]	Données de type caractères Unicode de longueur variable. n peut être compris entre 1 et 4 000. max indique que la taille de stockage maximale est de $2^{31}-1$ octets.	La taille de stockage, en octets, est le double du nombre de caractères entrés plus 2 octets
ntext	Données Unicode de longueur variable ne pouvant pas dépasser $2^{30} - 1$ caractères (c'est-à-dire 1 073 741 823).	La taille de stockage, en octets, est le double du nombre de caractères entrés

DRIOUCH B.

6

Types de Données

Chaînes binaires		
binary [(n)]	Données binaires de longueur fixe de n octets, où n est une valeur comprise entre 1 et 8 000	L'espace mémoire occupé est de n octets.
Varbinary[(n max)]	Données binaires de longueur variable. n est une valeur comprise entre 1 et 8 000. max indique que l'espace mémoire maximal occupé est de $2^{31}-1$ octets	La taille mémoire est la longueur réelle des données entrées, plus deux octets
image	Données binaires de longueur variable occupant de 0 à $2^{31} - 1$ (2 147 483 647) octets.	

Autres types de données	
cursor	timestamp
sql_variant	Uniqueidentifier
table	xml

DRIOUCH B.

7

Déclaration et Affectation des variables

■ Déclaration des variables locales

- Syntaxe générale : Declare @nom_variable type

Le caractère @ est obligatoire

- Exemple : Declare @maVariable int

- Déclaration multiple :

Declare @var1 type1, @var2 type2

- Exemple :Declare @x int, @y int, @z char

NB : Declare @x ,@y int est incorrecte

■ Affectation

Syntaxe générale : Select @variable=Expression où set @variable=Expression

- Exemple :select @i=3

set @j=4

select @str='TSDI'

- Affectation multiple

select @i=3,@j=4,@str='TSDI' est correcte

- Mais set @i=3, @j=4, @str='TSDI' est une affectation incorrecte.

DRIOUCH B.

8

Affichage

■ Affichage des valeurs

Pour afficher le contenu d'une variable on utilise la même instruction select.

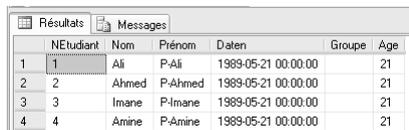
Select @i

Affichage multiple : Select @i,@j,@str

NB: On peut utiliser **select** pour affecter une valeur ou bien pour afficher une autre, mais pas pour faire les deux, donc l'instruction select @i=20, @str est incorrecte.

Affichage avec print : Print 'Chaine de caractère'

Et dans l'analyseur de requête SQL on a deux sortie d'affichage Messages pour Print et Table pour Select



	NEtudiant	Nom	Prénom	Dater	Groupe	Age
1	1	Ali	P-Ali	1989-05-21 00:00:00		21
2	2	Ahmed	P-Ahmed	1989-05-21 00:00:00		21
3	3	Imane	P-Imane	1989-05-21 00:00:00		21
4	4	Amine	P-Amine	1989-05-21 00:00:00		21



(4 ligne(s) affectée(s))
(4 ligne(s) affectée(s))

DRIOUCH B.

9

Exemple

Exemple de variable de type table :

```
Declare @stg table
  (numInsc int primary key,
  nom varchar(20),
  prenom varchar(20),
  moyenne numeric(4,2))
```

/*la particularité des variables de type table, est qu'on peut utiliser des commandes insert, select, update, delete */

```
insert into @stg values(103,'LAAROUSSI','SALAH',14)
insert into @stg values(107,'AADISSA','Youness',14.5)
insert into @stg values(200,'SOQRAT','Sanaa',12.5)
```

```
Select * from @stg
Select avg(moyenne) from @stg
```

DRIOUCH B.

10

Les Variables Globales

■ Les variables globales

Les variables globales sont affectées directement par le serveur, elle retournent une seule valeur, elle sont utilisées pour communiquer une information au client, elle sont notées @@nom_variable

Exemple :

@@error : indique le type d'erreur survenu lors de la dernière instruction.

@@rowcount : indique le nombre de lignes affectées par la dernière instruction.

@@identity : indique la valeur affecté à un attribut avec la propriété identity

■ Bloc d'instructions

Un bloc d'instruction est une ensemble d'instruction T-SQL qui sont considéré comme un tout (une seule).

Un bloc d'instruction peut contenir d'autres sous blocs.

Pour déclarer un bloc d'instructions en T-SQL en utilise :

Bloc Instruction

■ Bloc d'instructions

```
Begin
  --instruction(1)
  --instruction(2)
  ...
  --instruction(N)
end
- Example:
declare @i int,@j int
begin
  set @i=2
  set @j=3
  select @i=@i+1
  set @j=@j-1
  select @i as 'i', @j as 'j'
end
```

Structure Alternative

■ Structure Alternative

La structure alternative est une structure de contrôle qui permet d'exécuter un de deux actions suivant une condition. Syntaxe :

```
If(condition)
    -instruction ou bloc d'instruction
else
    -instruction ou bloc d'instruction
```

NB : la partie « else » est optionnelle. Il est possible d'imbriquer des if.

- Exemple :

```
Declare @stg table(numInsc int primary key, nom varchar(20), prenom
    varchar(20),moyenne numeric(4,2))
```

```
insert into @stg values(103,'LAAROUSSI','SALAH',14)
```

```
If not exists(select * from @stg )
```

```
    Print 'la table est vide'
```

```
Else
```

```
    Print 'la table n'est pas vide'
```

DRIOUCH B.

13

Structure Itérative

■ Structure Itérative

La structure itérative est une structure qui permet d'exécuter un même traitement plusieurs fois. Syntaxe générale :

```
While(condition)
    -instruction ou bloc d'instructions
où
```

```
Etiquette :
    -instruction ou bloc d'instructions
goto etiquette
```

- Exemple : calcule de la factorielle d'un nombre

DRIOUCH B.

14

Structure Itérative

```
Declare @i int, @f int,@n int
select @n=6, @f=1, @i=1
while (@i<=@n)
begin
    set @f=@f*@i
    set @i=@i+1
end
select @f as "le factoriel"
```

- Deuxième Solution :

```
Declare @i int, @f int, @n int
select @n=6, @f=1, @i=1
label:
    set @f=@f*@i
    set @i=@i+1
if(@i<=@n) goto label
select @f as "le factoriel"
```

DRIOUCH B.

15

Structure de Choix

■ Structure de choix (CASE)

La fonction CASE est une expression Transact-SQL spéciale qui permet l'affichage d'une valeur de remplacement en fonction de la valeur d'une colonne. Ce changement est temporaire. Par conséquent, aucune modification permanente n'est apportée aux données.

Cet exemple affiche dans le jeu de résultats d'une requête, le nom complet de la ville dans laquelle vit chaque Formateur :

SELECT nom, CASE ville WHEN 'CA' THEN 'Casablanca' WHEN 'Kn' THEN 'Kenitra' WHEN 'RB' THEN 'Rabat' END AS 'ville d'affectation' FROM Auditeur ORDER BY nom	SELECT nom, CASE WHEN note<8 THEN 'Éliminé' WHEN Note>=8 and Note<10 THEN 'Redoublement' WHEN Note>=10 THEN 'Admis' END AS 'Décisions' FROM Auditeur ORDER BY Décisions
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

DRIOUCH B.

16

Conversion de Type

■ Fonctions de conversion

Certaines conversions ne peuvent être automatiquement réalisées par le système. Nous devons alors réaliser ces conversions de manière explicite au moyen des fonctions de conversion `CAST(expression AS data_type[(length)])` et `CONVERT(data_type [(length)], expression [,style])`.

```
SELECT CONVERT(DATETIME,'10-14-2011',110) AS "Date au Format USA"
```

	Date au Format USA
1	2011-10-14 00:00:00

```
SELECT CONVERT(varchar,GetDate(),110) AS "Date au Format USA"
```

	Date au Format USA
1	10-14-2011

```
SELECT CONVERT(Decimal(10,3),sum(prix)) AS "TOTAL CA Net" FROM affectevol
```

	TOTAL CA Net
1	5400.000

```
SELECT Cast(Getdate() as varchar) as "Date Texte"
```

	Date Texte
1	oct 14 2011 9:03AM

DRIOUCH B.

17

Transactions

■ Traitement des transactions

Une transaction est une suite d'opérations effectuées comme une seule unité logique de travail. Une unité logique de travail doit posséder quatre propriétés appelées propriétés ACID (Atomicité, Consistance, Isolation et Durabilité).

- Atomicité : succès ou échec
- Consistance : tout est fait ou rien n'est fait
- Isolation : indépendant d'autres transactions ou événements
- Durabilité : les changements, une fois traités, ne peuvent pas être annulés

Syntaxes

Début de transaction : `BEGIN TRAN[SACTION] [nomtransaction]`

Validation de transaction : `COMMIT TRAN[SACTION] [nomtransaction]`

Annulation de transaction : `ROLLBACK TRAN[SACTION] [nomtransaction|nom P.C.]`

un point de contrôle (P.C.) : `SAVE TRAN[SACTION] [nom P.C.]`

Exemple :

```
begin TRAN a
    insert into Passe_ex values(1,3,12.3)
    save tran ab
    insert into Passe_ex values(1,5,12.3)
    rollback tran ab
    insert into Passe_ex values(2,3,12.3)
```

DRIOUCH B.

```
Commit Tran a
```

18

Transactions (Exp)

```
create database Employee
Go
use Employee
Go
Create table Employee(id int primary key,
  nom varchar(50), solde real)
Go
truncate table Employee
Go
Insert into Employee values(1, 'Ali',7000)
Insert into Employee values(2, 'Imane',0)
Insert into Employee values(4, 'Ahmed',0)
Insert into Employee values(5, 'Hanane',0)
Insert into Employee values(6, 'Khadija',0)
Go
Select * From Employee
Go
```

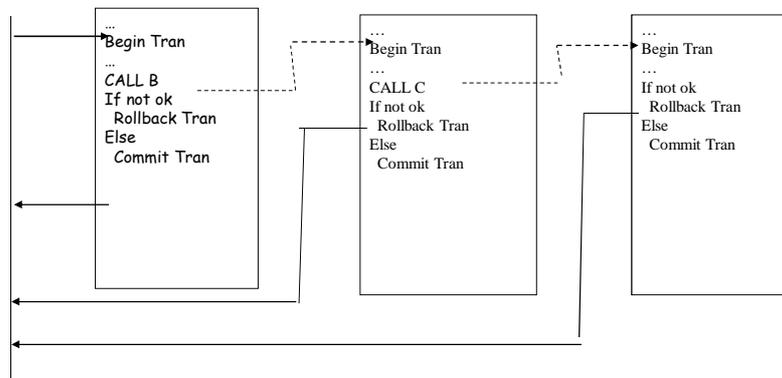
```
Begin tran a
declare @cpt int
set @cpt=0
Update Employee Set Solde=Solde-5000 where
id=1
set @cpt=@cpt+@@rowcount
Update Employee Set Solde=Solde+1000 where
id in(2,3,4,5,6)
set @cpt=@cpt+@@rowcount
if (@cpt=6)
  commit tran a
else
  rollback tran a
Go
Select * From Employee
Go
```

DRIOUCH B.

19

Transactions

■ Transactions imbriquées



DRIOUCH B.

20

Gestion des verrous

■ Gestion des verrous

Lors de transactions concurrentes, SQLServer gère automatiquement des verrous afin de garantir la cohérence des données de chaque transaction.

Une transaction ne pourra pas modifier des pages accessibles par une autre transaction, et ne pourra lire des pages en cours de modifications (lecture cohérente).

On peut agir sur les verrous de plusieurs façon, au niveau de la configuration et au niveau des transactions.

Mode de verrouillage	Description
Partagé	Protège une ressource en accordant uniquement l'accès en lecture. Aucune autre transaction ne peut modifier les données tant qu'il existe des verrous partagés sur la ressource.
Exclusif	Indique une modification de données, telle qu'une insertion, une mise à jour ou une suppression. Assure que plusieurs mises à jour ne puissent pas être effectuées simultanément sur la même ressource.
Mise à jour	Empêche une forme commune de blocage. Une seule transaction à la fois peut obtenir un verrou de mise à jour sur une ressource. Si la transaction modifie la ressource, le verrou de mise à jour est transformé en verrou exclusif.
Schéma	Utilisé lors de l'exécution d'une opération dépendant du schéma d'une table. Les types de verrous de schéma sont Modification du schéma (Sch-M) et Stabilité du schéma (Sch-S).
Intention	Établit une hiérarchie de verrous. Les types de verrous intentionnels les plus fréquents sont le verrou intentionnel de partage, le verrou intentionnel de mise à jour et le verrou intentionnel d'accès exclusif. Ces verrous indiquent qu'une transaction opère sur certaines (pas toutes) ressources inférieures de la hiérarchie. Les ressources de niveau inférieur auront un verrou Partagé, Mise à jour ou Exclusif.

DRIOUCH B.

Gestion des Erreurs

■ Messages d'erreur

```
USE master
```

```
GO
```

```
sp_addmessage @msgnum = 50005,  
              @severity = 10,  
              @msgtext = N'No think',  
              @lang='us_english';
```

```
go
```

```
sp_addmessage @msgnum = 50005,  
              @severity = 10,  
              @msgtext = N'RIEN',  
              @lang='french';
```

```
GO
```

```
--Utilisation de RAISERROR
```

```
RAISERROR (50005,10,1)
```

```
--demande l'enregistrement du message dans le journal des événements
```

```
RAISERROR(50005,16,1) With log
```

```
Go
```

```
sp_dropmessage @msgnum = 50005, @lang='all';
```

```
GO
```

DRIOUCH B.

22

Gestion Erreurs

■ Niveau de Gravité

Niveau de gravité	Description
0-9	Messages qui retournent des informations d'état ou qui signalent des erreurs sans gravité. Le Moteur de base de données ne dédenche pas d'erreurs système dont les niveaux de gravité sont compris entre 0 et 9.
10	Messages qui retournent des informations d'état ou qui signalent des erreurs sans gravité. Pour des raisons de compatibilité, le Moteur de base de données convertit le niveau de gravité 10 en 0 avant de retourner les informations d'erreur à l'application appelante.
11-16	Indique les erreurs pouvant être corrigées par l'utilisateur.
17-19	Indique des erreurs logicielles qui ne peuvent pas être corrigées par l'utilisateur. Informez votre administrateur système de l'existence du problème.
20-25	Indique l'existence de problèmes système et d'erreurs irrécupérables, ce qui signifie que la tâche du Moteur de base de données exécutant une instruction ou un lot n'est plus en cours d'exécution. La tâche enregistre des informations sur ce qui s'est produit, puis se termine. Dans la plupart des cas, la connexion de l'application à l'instance du Moteur de base de données se termine également.

DRIOUCH B.

23

Les Exceptions

■ les Exceptions:

TRY...CATCH :Implémente la gestion des erreurs pour Transact-SQL

```
BEGIN TRY
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

- ERROR_NUMBER() renvoie le numéro de l'erreur.
- ERROR_SEVERITY() renvoie la gravité de l'erreur.
- ERROR_STATE() renvoie le numéro d'état de l'erreur.
- ERROR_PROCEDURE() renvoie le nom de la procédure stockée ou du déclencheur dans lequel s'est produit l'erreur.
- ERROR_LINE() renvoie le numéro de ligne au sein de la routine qui a entraîné l'erreur.
- ERROR_MESSAGE() renvoie le texte complet du message d'erreur. Ce texte comprend les valeurs fournies pour tous les paramètres substituables, tels que les longueurs, les noms d'objet ou les heures.

DRIOUCH B.

24

Levée d'une Exception

- **Levée d'une erreur**
- Si une erreur survient dans votre code, vous pouvez prendre l'initiative pour cela, comme nous l'avons fait jusqu'à présent. Pour mieux personnaliser comment une exception est gérée lorsqu'elle survient, vous pouvez lever une erreur. Pour cela, Transact-SQL fournit la fonction **RAISERROR()**.
- Cette fonction accepte trois arguments requis :
 - le premier (une constante entière > 50000, un objet *msg_str*, une variable locale)
 - un numéro qui représente le niveau de gravité de l'erreur entre 0 et 25
 - état de l'erreur entre 1 et 127

DRIOUCH B.

25

Exception (Exp)

```
Begin tran a
BEGIN Try
Declare @cpt int
Set @cpt=0
if ((select solde from Employe where id=1)>=5000)
begin
Update Employe Set Solde=Solde-5000 where id=1
set @cpt=@cpt+@@rowcount
end
else
RAISERROR('Solde insuffisant',15,1)
Update Employe Set Solde=Solde+1000 where id in(2,3,4,5,6)
set @cpt=@cpt+@@rowcount
if (@cpt<>6)
RAISERROR('Montant Non répartie : Opération Annuler ',15,1)
Commit tran a
END Try
BEGIN Catch
Rollback tran a
select ERROR_MESSAGE() AS ErrorMessage
END Catch
```

DRIOUCH B.

26

Exercices

1. Factoriel de n.
2. Somme = $x^1/1! + x^2 / 2! + \dots + x^n/n!$ pour x et n.
3. Pour une valeur A, chercher le plus petit n qui vérifie $n! \geq A$.
4. Tableau de multiplication de N, dans un variable table.
5. Nombre premier $\leq N$.
6. PGDC de a et b.
7. Equation 2eme degré.

Solution sur le Forum du site :
<http://www.ista-ntic.net/>

DRIOUCH B.

27

Les Curseurs

Les curseurs permettent de réaliser des traitements itératifs sur des jeux de résultats, comme le balayage d'une table, enregistrement par enregistrement, en lecture seul.

▪ **Syntaxe**

DECLARE *cursor_name* CURSOR FOR *select_statement*

▪ **Arguments :**

cursor_name : Nom du curseur de serveur Transact-SQL défini. L'argument *cursor_name* doit respecter les conventions se rapportant aux identificateurs.

select_statement : Instruction SELECT standard qui définit le jeu de résultats du curseur. Les mots-clés COMPUTE, COMPUTE BY, FOR BROWSE et INTO ne sont pas autorisés dans l'instruction SELECT d'une déclaration de curseur.

- L'instruction OPEN remplit le jeu de résultats tandis que l'instruction FETCH renvoie une ligne à partir de ce jeu de résultats.
- Les autorisations DECLARE CURSOR sont octroyées par défaut à tout utilisateur qui a des autorisations SELECT sur les vues, les tables et les colonnes utilisées par le curseur.
- Le variable globale @@FETCH_STATUS établit un rapport d'état de la dernière instruction FETCH
 - 0 : L'instruction FETCH a réussi.
 - 1 : L'instruction FETCH a échoué ou la ligne se situait au-delà du jeu de résultats.
 - 2 : La ligne recherchée est manquante.

DRIOUCH B.

28

Les Curseurs (Exp)

■ **Exemple:**

```
BEGIN
Declare @NuAud int, @nom varchar(20)
DECLARE Auditeur_cursor CURSOR FOR SELECT NuAud, nom FROM Auditeur

OPEN Auditeur_cursor
FETCH NEXT FROM Auditeur_cursor INTO @NuAud, @nom

While @@FETCH_STATUS=0
begin
    print 'Num : ' + Cast(@NuAud as varchar(20)) + ' - Nom: ' + @nom
    FETCH NEXT FROM Auditeur_cursor INTO @NuAud, @nom
End
CLOSE Auditeur_cursor
DEALLOCATE Auditeur_cursor
GO
END
```

```
Num : 1 - Nom: Ali
Num : 2 - Nom: Ahmed
Num : 3 - Nom: Karim
```

DRIOUCH B.

29

Les Curseurs (SCROLL)

■ **Syntaxe**

```
DECLARE cursor_name SCROLL CURSOR FOR select_statement
Open cursor_name
FETCH [[NEXT|PRIOR|FIRST|LAST |
      ABSOLUTE {n|@nvar} |
      RELATIVE {n|@nvar}]
FROM cursor_name INTO @var...
Close cursor_name
DEALLOCATE cursor_name
```

Next: suivant

Prior: avant

First: premier

Last: derner

Absolute n: position n

Relative n: avancé par n position (si n négatif reculé de n position)

DRIOUCH B.

30

Les Curseurs (SCROLL) (Exp)

```
Declare @NuAud int, @nom varchar(20)
DECLARE Auditeur_cursor SCROLL CURSOR FOR SELECT NuAud, nom FROM Auditeur
OPEN Auditeur_cursor
```

```
FETCH NEXT FROM Auditeur_cursor INTO @NuAud, @nom
if @@FETCH_STATUS=0
    print '1 Num : ' + Cast(@NuAud as varchar(20)) + ' - Nom: ' + @nom
```

```
FETCH absolute 3 FROM Auditeur_cursor INTO @NuAud, @nom
if @@FETCH_STATUS=0
    print '2 Num : ' + Cast(@NuAud as varchar(20)) + ' - Nom: ' + @nom
```

```
FETCH relative -1 FROM Auditeur_cursor INTO @NuAud, @nom
if @@FETCH_STATUS=0
    print '3 Num : ' + Cast(@NuAud as varchar(20)) + ' - Nom: ' + @nom
```

```
FETCH last FROM Auditeur_cursor INTO @NuAud, @nom
if @@FETCH_STATUS=0
    print '4 Num : ' + Cast(@NuAud as varchar(20)) + ' - Nom: ' + @nom
CLOSE Auditeur_cursor
DEALLOCATE Auditeur_cursor
```

DRIOUCH B.

31

Exercices

- Soit les table suivantes(DB_Calcul):
 - Pair_impair (num, reponse): écrire un programme qui modifie réponse par pair ou impaire selon valeur de num
 - Premier(num, reponse): écrire un programme qui modifie réponse par premier ou non selon valeur de num
 - Calcul(Num1,Num2,Op,Resultat): écrire un programme qui calcul le Resultat selon Num1, Num2 et Op(+,-,*,/)
- Soit le schéma relationnel (GestStg) suivant:
 - Stagiaire(**IdStg**, Nom, Moyenne)
 - Matiere(**IdMat**, Libelle, Coeff)
 - Note(**IdStg**, **IdMat**, Note)

Écrire un programme qui met à jour la moyenne de chaque stagiaire, sans affichage détailler (affichage à partir de la table stagiaire avec select)

DRIOUCH B.

32

Procédures Stockées

Une procédure stockée (Stored Procedure pour SQL Server) est une suite d'instructions SQL stockées dans la base de données et pouvant être exécutée par appel de son nom avec ou sans paramètre.

- Les procédures stockées diffèrent des instructions SQL :
 - Leur syntaxe est vérifiée et elles sont compilées. C'est le code compilé qui est utilisé lors des appels
 - Ne sont pas appelées automatiquement, mais doivent faire l'objet d'un appel explicite de la part de l'utilisateur.
 - Peuvent être appelées par plusieurs applications frontales
- Avantages :
 - Améliorent les performances par utilisation du code compilé
 - Renforcent l'intégrité de la base : en centralisant les traitements en un endroit unique → unicité du code

DRIOUCH B.

33

Procédures Stockées

Syntaxe

```
CREATE PROC[EDURE] procedure_name  
[ { @parameter data_type }  
[ VARYING ] [ = default ] [ OUTPUT ]  
] [ ,...n ]  
[ WITH  
{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]  
[ FOR REPLICATION ]  
AS sql_statement [ ...n ]
```

- @parameter : Un paramètre de la procédure.
- Default : Valeur par défaut pour le paramètre. La valeur par défaut peut contenir des caractères génériques (% , _ , [] et [^]) si la procédure utilise le nom du paramètre avec le mot clé LIKE
- OUTPUT: Indique que le paramètre est un paramètre de retour renvoyé par la procédure.

DRIOUCH B.

34

Procédures Stockées

- {RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}
RECOMPILE indique que SQL Server n'utilise pas le cache pour le plan de cette procédure et que la procédure est recompilée à l'exécution. Utilisez l'option RECOMPILE lorsque vous utilisez des valeurs temporaires ou atypiques sans remplacer le plan d'exécution placé en mémoire cache.
- ENCRYPTION
indique que SQL Server crypte l'entrée de la table syscomments contenant le texte de l'instruction CREATE PROCEDURE. L'utilisation de l'argument ENCRYPTION évite la publication de la procédure dans le cadre de la réplication SQL Server.
- FOR REPLICATION
Indique que les procédures stockées créées pour la réplication ne peuvent pas être exécutées sur l'abonné. Une procédure stockée créée avec l'option FOR REPLICATION est utilisée comme filtre de procédure stockée et n'est exécutée que pendant la réplication. Cette option ne peut pas être utilisée avec l'option WITH RECOMPILE.
- La taille maximale d'une procédure stockée est limitée à 128 Mo.
- Une procédure stockée peut retourner une valeur statique entier pour indiqué son état d'exécution : if condition RETURN -1
- Pour supprimer une PS : Drop Proc procedure_name
- Pour Modifier une PS : Alter Proc procedure_name paramètre as sql_stat

DRIOUCH B.

35

Procédures Stockées

▪ Exemple : Factoriel

```
Use DB_Calcul
Go
Create Proc Factoriel @n int
as
begin
    declare @f int, @i int
    select @f=1, @i=1
    while (@i<=@n)
        begin
            set @f=@f*@i
            set @i=@i+1
        end
    select @n as "N", @f as "Factoriel de N"
End
Go
```

Appel d'une Procédure Stockée

```
Exec Factoriel 5
Go
ou
Exec Factoriel @n=5
Go
```

	N	Factoriel de N
1	5	120

DRIOUCH B.

36

Procédures Stockées

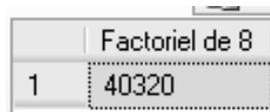
■ Exemple : FactorielS, Param. Sortie

```
Create Proc FactorielS @n int,  
@f int OUTPUT  
as  
Begin  
declare @i int  
select @f=1, @i=1  
while (@i<=@n)  
begin  
set @f=@f*@i  
set @i=@i+1  
end  
End  
Go
```

DRIOUCH B.

■ Appel de la PS.

```
Declare @fact int  
Set @fact=0  
Exec FactorielS 8, @fact OUTPUT  
Select @fact as "Factoriel de 8"  
Go
```



Factoriel de 8	
1	40320

37

Procédures Stockées

■ Exemple

Sur la base auditeur en ajoute une colonne moyenne, on crée une procédure stocké pour le résultat de la moyenne des examens que l'auditeur a passé.

-- Ajout de la colonne moyenne

```
alter table auditeur  
add moyenne real constraint ct_my check (moyenne between 0 and 20)
```

```
Create PROC CalcMy  
as  
begin  
update auditeur set moyenne=(select avg(note) from Passe_Ex Where  
passe_ex.nuAud=auditeur.NuAud)  
End  
Go  
Exec CalcMy
```

DRIOUCH B.

38

Procédures Stockées

■ Exemple avec paramètre

procédure qui fait le même calcul, mais pour un seul auditeur, son numéro est passé en paramètre.

```
Create PROC CalcMyP @Aud int= null
as
Begin
if @Aud is null
    update auditeur set moyenne=(select avg(note) from Passe_Ex Where
    passe_ex.nuAud=auditeur.NuAud);
else
    update auditeur set moyenne=(select avg(note) from Passe_Ex Where
    passe_ex.nuAud=auditeur.NuAud) where auditeur.NuAud=@Aud;
End
Go
Exec CalcMyP 2
Exec CalcMyP @Aud=2
Exec CalcMyP
DRIOUCH B.
```

39

Procédures Stockées

■ Exemple avec valeur de retour d'état

```
Create Proc Factoriel @n int, @fact int output as
Begin
declare @f int, @i int
select @f=1, @i=1
if @n<0
    return -1
else
    begin
        while (@i<=@n)
        begin
            select @f=@f*@i, @i=@i+1
        end
        set @fact=@f
        return 0
    end
end
Go
Declare @resultat int, @etat int
Exec @etat=Factoriel -2,@resultat output
Select @etat as "Etat", @resultat as "Resultat"
DRIOUCH B.
```

40

Exercices

Fonctions

```
CREATE FUNCTION [ schema_name. ] function_name (  
[ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type [=default] } [ ,...n ] ] )
```

- **Scalar Functions (retourne une valeur)**

```
RETURNS return_data_type [ WITH <function_option> [ ,...n ] ]  
[ AS ]  
BEGIN  
    function_body  
    RETURN scalar_expression  
END [ ; ]
```

- **Inline Table-valued Functions (retourne une table inline)**

```
RETURNS TABLE [ WITH <function_option> [ ,...n ] ]  
[ AS ]  
RETURN [ ( ) select_stmt [ ] [ ; ]
```

- **Multistatement Table-valued Functions (retourne une table on multi-instruction)**

```
RETURNS @return_variable TABLE < table_type_definition >  
[ WITH <function_option> [ ,...n ] ]  
[ AS ]  
BEGIN  
    function_body  
    RETURN  
END [ ; ]
```

NB : Les fonctions ne permettent pas les instructions qui produisent un effet secondaire, tel que la modification d'une table (update)

Pour la suppression et la modification : Drop Function F, Alter Function F

Fonctions

■ Exemple (Une Fonction Scalaire):

```
CREATE FUNCTION Factoriel (@n int)
RETURNS bigint
AS
BEGIN
    declare @f bigint, @i int
    select @f=1, @i=1
    while (@i<=@n)
    begin
        select @f=@f*@i, @i=@i+1
    end
    RETURN @f
END
Go
Select dbo.Factoriel(6) as Factoriel;
ou
Declare @r bigint
Set @r=dbo.Factoriel(15)
Print cast(@r as varchar)
DRIOUCH B.
```

43

Fonctions

■ Exemple : une fonction table online

```
USE GestStg
--Drop Function NoteStg
CREATE FUNCTION NoteStg (@idstg int)
RETURNS Table
AS
RETURN (Select libelle, avg(note) as NoteM from Matiere inner join note on
Matiere.idmat=note.idmat where idstg=@idstg group by libelle);
Go
Select Libelle, NoteM from dbo.NoteStg(2) where NoteM>=10;
Go
DRIOUCH B.
```

44

Fonctions

■ Exp : une fonction table à instructions multiples

```
Create FUNCTION NoteStg2 (@idstg int)
RETURNS @Resultat Table (Matiere varchar(50), Note decimal(4,2))
AS
begin
    insert into @Resultat Select 'Nom : ' + nom, Null from stagiaire where idstg=@idstg
    insert into @Resultat Select libelle, avg(note)
        From Matiere inner join note on Matiere.idmat=note.idmat
        Where idstg=@idstg Group by Matiere.idmat, libelle
    insert into @Resultat select 'Moyenne :', Moyenne from stagiaire where idstg=@idstg
RETURN
end
Go

Select * from dbo.NoteStg2(1);
Go
```

DRIOUCH B.

45

Exercices

- Ecrire une fonction qui retourne la moyenne pour un stagiaire dont id donnée en paramètre
- Écrire une fonction qui retourne pour un stagiaire donnée, la liste des modules avec la moyenne des notes pour chaque module.

DRIOUCH B.

46

Triggers (Déclencheurs)

Un triggers est une Forme évoluée de règles utilisées pour renforcer l'intégrité de la base de données, on peut dire aussi des contraintes d'intégrité personnalisés.

- Les triggers sont un type particulier de procédure mémorisée.
 - sont attachés à des tables
 - réagissent aux fonctions de création (Insert), modification (Update) et suppression (Delete)
 - ne peuvent pas être appelés explicitement dans les applications
- Les triggers sont déclenchés automatiquement par le noyau SQL à chaque intervention sur la table qui les supportent.
- Un trigger est toujours associé à une table, qui peut avoir au maximum trois triggers pour (Insertion, modification et suppression de ligne)
- La suppression d'une table entraîne la destruction de ses triggers
- Avec SQL 2005 et plus, on peut avoir des déclencheurs sur LDD (Langage de Définition de Donnée) comme (Create, Alter, Drop) et à certaines procédures stockées système qui effectuent des opérations de type LDD

Triggers (Déclencheurs)

■ Principe de fonctionnement.

Deux tables virtuelles (variable local temporaire dans le programme) sont créées au moment de la MAJ des données sur une table (INSERTED, DELETED) en lecture seule. Elles sont destinées à contenir les lignes de la table sur lesquelles ont été effectuées des opérations.

Les tables INSERTED et DELETED peuvent être utilisées par le trigger pour déterminer comment le traitement doit se dérouler. Ce traitement est à écrire par le développeur.

- Cas de suppression d'une ligne de table (Delete) : La/les lignes supprimées sont placées dans la table temporaire DELETED et supprimées de la table réelle;
- Cas de création d'une ligne de table (Insert) : La/les lignes nouvelles sont placées dans la table temporaire INSERTED et dans la table réelle;
- Cas de modification d'une ligne de table (Update) : La/les lignes avant modification sont placées dans la table temporaire DELETED et la/les lignes après modification sont placées dans la table temporaire INSERTED et dans la table réelle.

Triggers (Déclencheurs)

- Principe de fonctionnement.
 - INSTEAD OF (au lieu de): Les déclencheurs INSTEAD OF peuvent être définis sur des tables ou des vues; remplace le traitement de l'action (Insert, Delete ou Update), en peut avant l'action, est utilisé comme suit:
 - Cas de suppression (Delete) : La/les lignes supprimées sont placées dans la table temporaire DELETED et non supprimées de la table réelle;
 - Cas de création d'une (Insert) : La/les lignes nouvelles sont placées dans la table temporaire INSERTED et non dans la table réelle;
 - Cas de modification d'une ligne de table (Update) : La/les lignes avant modification sont placées dans la table temporaire DELETED et reste dans la table réelle, et la/les lignes après modification sont placées dans la table temporaire INSERTED et non dans la table réelle.

Triggers (Déclencheurs)

■ Syntaxe :

```
CREATE TRIGGER nom_trigger ON nom_table FOR INSERT  
AS
```

```
    bloc d'instruction SQL
```

```
CREATE TRIGGER nom_trigger ON nom_table FOR UPDATE  
AS
```

```
    bloc d'instruction SQL
```

```
CREATE TRIGGER nom_trigger ON nom_table FOR DELETE  
AS
```

```
    bloc d'instruction SQL
```

ou

```
CREATE TRIGGER nom_trigger ON nom_table  
FOR INSERT, UPDATE  
AS
```

```
    bloc d'instruction SQL
```

Suppression d'un triggers : DROP TRIGGER nom_trigger

Modifier un triggers : la même syntaxe de Create, en remplaçant Create par Alter

Triggers

- **Exemple :**

Pour implémenté la contraint: de limiter le nombre de note par stagiaire par Matière à 3, sur la base GestStg, on doit utilisé les triggers:

```
CREATE TRIGGER LimitNote ON note
FOR INSERT
AS
begin
    if (select count(*) from note,inserted where note.idstg=inserted.idstg and
note.idmat=inserted.idmat)>3
        begin
            RAISERROR('nombre limite 3 notes par matièer, insertion annuler',15,1)
            rollback tran
        end
end
Go
```

DRIOUCH B.

51

Triggers

- **Exemple (Instead of):**

Pour la suppression d'une Matière:

```
CREATE TRIGGER Supp
ON Matière
INSTEAD OF DELETE
AS
begin
    Delete Form Note Where IdMat in (select idmat from deleted)
    Delete From Matière Where IdMat in (select idmat from deleted)
end
Go
```

Ici la suppression des notes ce passe avant la suppression des matières pour évité l'erreur provoqué par la contraint d'intégrité référentiel.

DRIOUCH B.

52

Exercice

- Sur la base GestStg:

Stagiaire (idstg, nom, moyenne)

Matiere (idmat, libelle, coeff)

Note (idstg, idmat, note)

- 1) Créer un trigger qui affiche les lignes inséré pour insert sur la table Matière.
- 2) Refaire l'implémentation des contraintes d'intégrité suivant par des triggers:
 - i. Référentiel (clé étranger)
 - ii. De domaine ($0 \leq \text{Note} \leq 20$)
- 3) Implémenté la suppression en cascade
- 4) Recalculer la moyenne pour chaque modification dans les notes