

Chapitre II : La programmation en SQL

1-Déclaration de variables

Le mot **DECLARE** Permet de déclarer les variables dans le corps d'une procédure.

Syntaxe: DECLARE @nomvar type

Exemple: DECLARE @msg varchar(100)

Affectation de valeurs :

Le mot **SET (ou select)** Permet d'alimenter le contenu d'une variable

Syntaxe :

SET @nomvar = expression

ou

Select @nomvar = expression

Exemple :

declare @a float

SET @a = (select avg(NOTE) from notes)

Ou

select @a=(select avg(NOTES) from notes)

5-L'instruction de Sortie :

Le mot **Print** permet d'afficher les messages et les valeurs des Variables alimentée dans une procédure.

Syntaxe :

Print 'message'

Print @variable

Remarque : si vous voulez afficher un message contient une valeur numérique, il faut convertir cette valeur en chaîne à l'aide de la méthode **Convert** (Type, @variable)

Syntaxe :

print 'la moyenne est : ' + convert(varchar(10),@a)

Les commentaires :

L'instruction **/*...*/** Permet de commenter la procédure par un texte qui ne sera pas pris en compte lors de l'exécution. Pour une seule ligne, utiliser **--**

Encadrer un Bloc d'instructions :

L'instruction BEGIN...END permet d'encadrer un bloc d'instructions constituant un groupe au moment de l'exécution.

Syntaxe

```
BEGIN
    Groupe d'instructions SQL
END
```

6- Les instructions Conditionnelles :

a- IF...ELSE

Permet de conditionner l'exécution de certaines instructions

Syntaxe

```
IF condition
    Groupe d'instructions SQL si la condition vraie
ELSE
    Groupe d'instructions SQL si la condition fausse
```

Exemple :

Si le salaire d'un salarié est inférieure à 3000, augmentation de 25 % du salaire.
Dans le cas contraire, l'augmentation sera de 10%.

@code=25

```
IF (select salaire from salarie
    where Cod = @code) < 3000
    BEGIN
        update salarie set Salaire = Salaire * 1.25
        where cod=@code
    END
ELSE
    BEGIN
        update salarie set Salaire = Salaire * 1.10
        where cod=@code
    END
```

Déclaration/Affectation/Affichage Multiple

Declare @x int, @y int, @z char

Attention :

Declare @x ,@y int est incorrecte.

affectation d'une variable

select @i=3 ;

set @j=4 ;

select @str='TSDI'

Exemple : Affectation multiple

select @i=3,@j=4,@str='TSDI' ;

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

Affichage:

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

Exemples : Select @i ;

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

Remarque 1 : 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.

Remarque 2 : Affichage avec print

On peut utiliser la commande print pour afficher un résultat sous format message. Syntaxe : Print 'Chaine de caractère'

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

Déclaration/Affectation/Affichage Multiple

Exemple de variable de type table :

```
Declare @stg table (numInsc int primary key, nom varchar(20), moyenne float)
```

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

```
insert into @stg values(103,'Jalimi',12.2),(107,'ibrahimi', 15.06)
```

```
insert into @stg values(200,'SOQRAT',10.89)
```

```
Select * from @stg
```

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

Les boucles :

GOTO etiquette: Permet un branchement inconditionnel à une étiquette.

```
DECLARE @index int
SET @index = 0
debut:
PRINT 'Index=' + CONVERT(varchar(2),@index)
SET @index = @index + 1
IF @index != 10
    GOTO debut
```

La boucle WHILE :

Permet d'exécuter un bloc d'instruction tant qu'une condition est réalisée.

Syntaxe :

```
WHILE condition
    Instruction ou bloc d'instructions SQL
```

Exemple :

```
Set @index=1
While @index<=10
Begin
    Print @index
    Set @index=@index+1
End
```

Le mot BREAK :

Permet de sortir d'une boucle **WHILE** même si les conditions de fin ne sont pas réalisées.

Exemple Factoriel:

Exemple : calcule de la factorielle d'un nombre

```
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"
```

2^{ème} Méthode: Goto:

```
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"
```

Les instructions Conditionnelles :

SYNTAXE de CASE:

CASE

- WHEN CONDITION 1 THEN resultat1
- WHEN CONDITION 2 THEN resultat2
- WHEN CONDITION N THEN resultat N
- ELSE resultat

END;

L'instruction CASE, permet d'attribuer des valeurs en fonction d'une condition

Remarque : 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.

Exemple : afficher le nom complet de la ville :

```
SELECT nom, CASE ville
  WHEN 'CA' THEN 'Casablanca'
  WHEN 'Kn' THEN 'Kenitra'
  WHEN 'RB' THEN 'Rabat'
END
FROM client ORDER BY nom ;
```

La fonction SQL CONVERT, permet de convertir une donnée d'un type en un autre, de façon semblable à [CAST\(\)](#).

La syntaxe est la suivante :

```
CONVERT ( type, expression );
```

```
CONVERT (Data_Type, (length), expression/value, style)
```

La syntaxe de la fonction CAST est la suivante :

```
SELECT CAST( expression AS type );
```

```
SELECT CAST('08/24/2020' AS DATE) AS DateStmp2;
```

exercice: Calculer

Somme= $x^1/1! + x^2/2! + \dots + x^n/n!$ Pour x et n .

```
DECLARE @x FLOAT, @n int, @f bigint, @som real, @i int
```

```
Select @x=3.5, @n=5, @i=1; @f=1, som=0
```

```
WHILE (@i <= @n)
```

```
BEGIN
```

```
Set @f=@f*@i
```

```
SET @som= @som + POWER(@x, @i) / @f
```

```
SET @i = @i + 1;
```

```
END
```

```
PRINT 'Le résultat de la série pour x = ' + CAST(@x AS VARCHAR(10)) + '  
et n = ' + CAST(@n AS VARCHAR(10)) + ' est : ' + CAST(@Somme AS  
VARCHAR(50));
```

```
GO
```

```
Ou Select 'la Somme: ' as " ", Convert(decimal(10,2), @som)
```

-
1. Créer une base de données avec la table client (idcl, nom, prenom, ville) , l'id du client est auto incrémenté.
 2. Insérer des lignes à la table client tant que le nombre de lignes déjà insérée est inférieure à 6.
 3. Afficher les clients avec leurs villes en majuscules
 4. -Afficher le nom et le prenom des clients de safi s'il existent et 'aucun client de safi' sinon

```
create database Client
```

```
use Client
```

```
–Q1:
```

```
create table client (idc int identity primary key,  
nom varchar (20),  
prenom varchar (20),  
ville varchar (20) );
```

```
–Q2:
```

```
declare @i int
```

```
set @i = 1
```

```
while (@i <= 6 )
```

```
begin
```

```
insert into client values ('N'+ CAST (@i as varchar ),'P' + CAST (@i as  
varchar ),'V'+ CAST (@i as varchar ) )
```

```
set @i = @i+1
```

```
end
```

```
— Pour réinitialiser la valeur d'une colonne d'identité Identity on utilise :  
dbcc checkident ('client',reseed,0)
```

```
–Q3 :
```

```
select nom,prenom,UPPER (ville) as ville_en_majuscule from client
```

```
–Q4 :
```

```
if exists (select * from client where ville = 'Safi')
```

```
select nom , prenom from clietn where ville = 'Safi'
```

```
else
```

```
print 'aucun client de safi'
```

Chapitre III : Les Curseurs

1 Qu'est-ce qu'un curseur

Dans tout ce qui a précédé, nous avons obtenu un résultat global, c-à-d Une instruction **select** renvoie zéro ou plusieurs ligne. Nous n'avons pas pu faire de traitement ligne par ligne.

Pour pouvoir exécuter d'autres commandes entre chaque ligne, nous utiliserons un **curseur**.

Le curseur est une variable de type **CURSOR** qui permet aux applications de manipuler le jeu de résultats ligne par ligne.

Il existe différentes étapes à suivre pour utiliser un curseur :

- Déclaration et ouverture de curseurs.
- Extraction de données à l'aide de curseurs (Le parcours du curseur).
- Fermeture et libération d'un curseur.

1 DECLARE CURSOR

Un curseur est une variable d'un type particulier CURSOR. Il se déclare donc à l'aide du mot clé DECLARE.

Syntaxe :

DECLARE MonCurseur CURSOR FOR REQUÊTE

Exemple: Déclarer un **curseur** MonCR qui permet d'accéder à toutes les colonnes de la table Eleve qui ont une Note >= 10.

DECLARE MonCR CURSOR FOR SELECT * from FROM Eleve WHERE Note>= 10

2 Ouverture de curseurs (Mot OPEN)

L'instruction **OPEN** déclenche l'exécution de la requête SELECT et charge le résultat.

Syntaxe

OPEN Mon_Cursor

3 Lecture de lignes (Le Mot FETCH)

L'instruction FETCH charge les valeurs de la ligne dans la liste de variables précisée en complément au niveau du mot clé **INTO**.

Syntaxe : FETCH Next from Mon_Cursor INTO @Variable;

4- L'instruction @@FETCH_STATUS

Renvoie l'état de la dernière instruction FETCH effectuée sur un curseur actuellement ouvert par la connexion.

Valeurs renvoyées par l'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.

Syntaxe: @@FETCH_STATUS = 0

Exemple:

```
While @@FETCH_STATUS = 0
```

```
Begin
```

```
    Instructions
```

```
end
```

5- Fermeture du curseur (Le MotCLOSE)

Pour Ferme le curseur il faut utiliser le mot close.

Syntaxe : CLOSE Mon_Cursor

6- Libérer la mémoire allouée (Le Mot DEALLOCATE)

Pour Libérer la mémoire allouée au curseur il faut utiliser le mot DEALLOCATE

Syntaxe : DEALLOCATE Mon_Cursor

Exemple Complet :

Pour obtenir la liste des matières ayant une Note >= 3

```
declare @m nchar(30)
```

```
DECLARE Mon_Cursor CURSOR FOR
```

```
    SELECT codem from mat Where cof>=3;
```

```
    OPEN Mon_Cursor ;
```

```
    FETCH NEXT FROM Mon_Cursor into @m
```

```
    WHILE @@FETCH_STATUS = 0
```

```
    BEGIN
```

```
        PRINT 'le code de la matière est : ' + @m
```

```
        FETCH NEXT FROM Mon_Cursor into @m
```

```
    END
```

CLOSE Mon_Cursor

DEALLOCATE Mon_Cursor

SCROLL CURSOR

Syntaxe :

```
DECLARE cursor_name SCROLL CURSOR FOR select_statement
Open cursor_name
```

```
FETCH [INEXT|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: dernier

Exemple:

```
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
```


SCROLL CURSOR

Exemple:

```
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
```