

## Bases de Données Avancées

### BDA

# Chapitre I : 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
```

## **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.

## **Chapitre II : 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
```