ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

MICROSOFT SQL SERVER

Prof. Maria EL HAIBA

Docteur en Informatique

Email: ***m.elhaiba@emsi.ma***

INTRODUCTION AU LANGAGE TRANSACT-SQL

- **Éléments du langage T-SQL (Variables, déclaration, affectation,…)**
- **Les structures de contrôle**
- Les curseurs, Les transactions
- Les procédures stockées, Les fonctions système
- La gestion des exceptions, Les déclencheurs

Définition Transact SQL

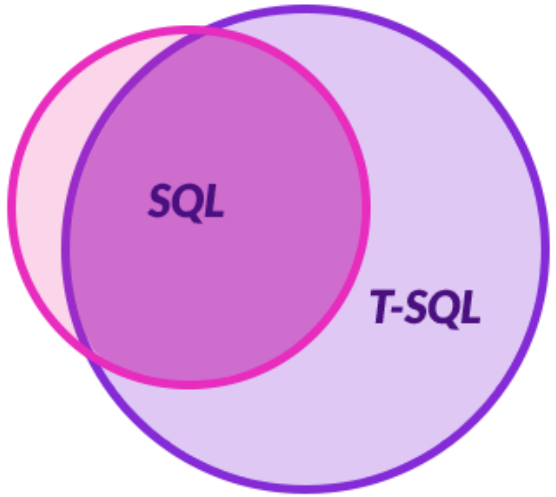
- La plupart des SGBDs relationnels offrent une **extension du SQL**, en y ajoutant des déclarations de variables et des structures de contrôles pour **améliorer leurs performances** ;

- **Transact-SQL** (ou encore appelé T-SQL) est en effet le monopole de Microsoft, qui est utilisé dans SQL Server. C'est un langage de programmation **procédural** permettant d'étendre les capacités de SQL par des fonctionnalités de bases de données que SQL seul ne peut implémenter.

Lien utile : Transact-SQL Reference

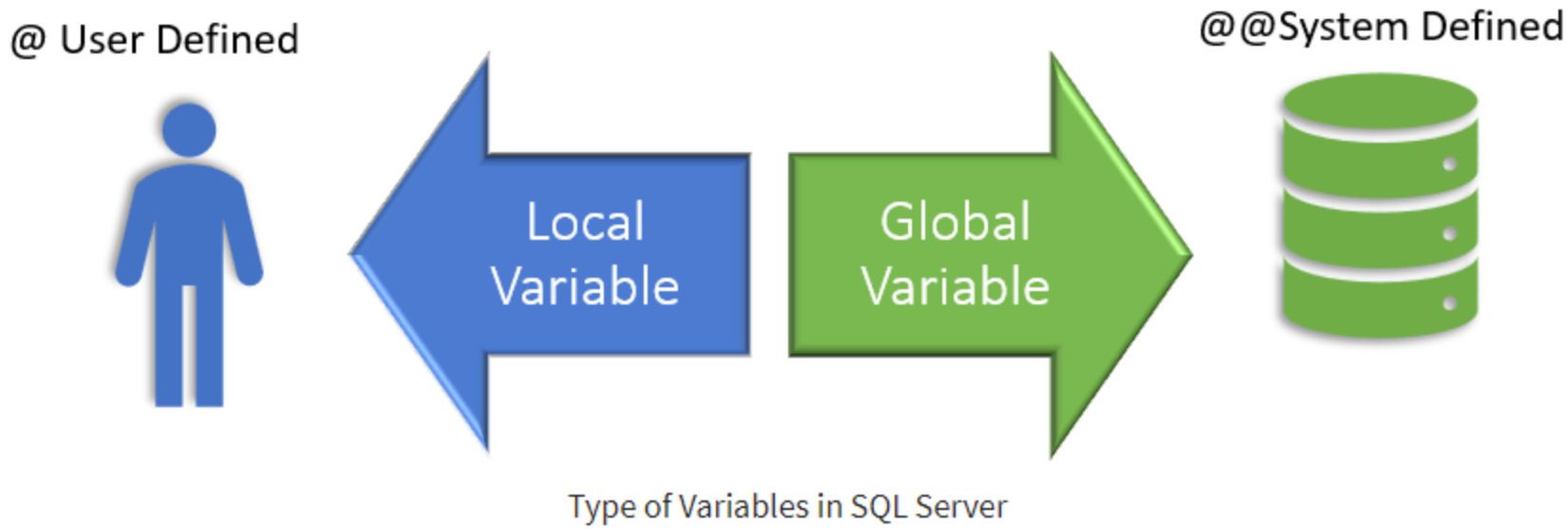
SQL vs T-SQL

T-SQL est une **implémentation évoluée de SQL** qui définit une batterie complète de toutes les opérations exécutables sur une base de données (lecture de données, opérations d'administration du serveur, ajout, suppression et mises à jour d'objets SQL - tables, vues, **transactions, procédures stockées, déclencheurs**, types de données personnalisés …-).



Éléments du langage T-SQL

- **Les variables et leurs types**



Type of Variables in SQL Server

Éléments du langage T-SQL

Les variables (locales) et leurs déclarations

- Dans Transact SQL, on utilise le mot réservé **DECLARE** pour déclarer des variables
- Les noms de variables (**locales**) sont précédés du **symbole @**
- Les **types de variables**, sont les **types SQL** (int, float, datetime, varchar,…)
- Les variables peuvent être **initialisées** avec des valeurs en utilisant la **fonction SET**

Exemple:

```
DECLARE
@Age int ;
SET @Age =30;
```

Affectation de valeur avec le mot réservé SET

- Un seul DECLARE est suffisant si vos déclarations sont suivies par une virgule (déclarer plusieurs variables en même temps)
- La dernière variable déclarée se termine par un point virgule (;)

Exemple:

```
DECLARE
@variable int = 12,
@nom varchar(30) ='Patoche',
@date DATE ='2021-05-21';
```

Affectation de valeur à l'initialisation

Éléments du langage T-SQL

- **Les variables globales du système**
  - Ensemble de variables **définies par le SGBD**
  - Elles sont précédées du **symbole @@**

Variable	Description
@@connections	Nombre de connexions actives
@@error	Code de la dernière erreur rencontrée (0 si aucune)
@@fetch_status	État d'un curseur lors de la lecture (0 si lecture proprement exécutée)
@@identity	Dernière valeur insérée dans une colonne auto incrémentée pour l'utilisateur en cours
@@max_connections	Nombre maximums d'utilisateurs concurrents
@@procid	Identifiant de la procédure stockée en cours
@@rowcount	Nombre de lignes concernées par le dernier ordre SQL
@@servername	Nom du serveur SGBDR courant
@@spid	Identifiant du processus en cours
@@trancount	Nombre de transactions en cours

Éléments du langage T-SQL

Structure d'un programme T-SQL

- T-SQL est organisé par un **bloc d'instructions** qui constitue un **groupe** au moment de l'exécution
- Un bloc commence par le mot réservé **BEGIN** et finit par le mot **END**
- La fonction **PRINT** peut être utilisé pour afficher le contenu d'une variable (ou bien utiliser la clause **SELECT**)

Exemple:

```
USE BD_Employes
DECLARE
@salaire_min MONEY
BEGIN
SELECT @salaire_min = MIN(SALAIRE) FROM Employes;
PRINT @salaire_min;
END;
```

Affectation de valeur avec une requête (Select)

Qu'il est aussi possible de faire : `SET @salaire_min =(SELECT MIN(SALAIRE) FROM Employes);`

Éléments du langage T-SQL

- **Exemples : Déclaration, affectation et affichage**

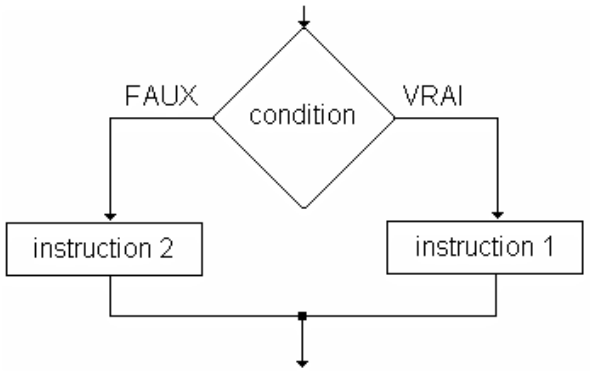
```
DECLARE --Déclaration de variables
@var_name varchar(30), @var int =23;
set @var_name='bonjour'; --Affectation d'une valeur
print @var_name --Affichage de la valeur
print @var;

/*Affectation et affichage avec provenance d'une table*/
DECLARE
@Prix int, @var1 int, @var2 int,  @var3 int;
SELECT @Prix = min(PUArt) FROM Article; --Affectation
PRINT @Prix; --Affichage par le PRINT

SELECT @var1= PUArt FROM Article WHERE NumArt= 1; --La requête ne doit produire qu'une ligne
SELECT @var2= PUArt FROM Article; --Dans ce cas, seule la dernière ligne est prise en compte
SELECT @var2 AS PUArt; --Affichage par la clause SELECT
SET @var3= (SELECT max(PUArt) FROM Article); --Affectation avec le mot SET
PRINT @var3;
```

Structures de contrôle

- Les structures de contrôle permettent de choisir la façon dont les différentes instructions vont être exécutées.
- Les **importantes** structures de contrôle en T-SQL sont :

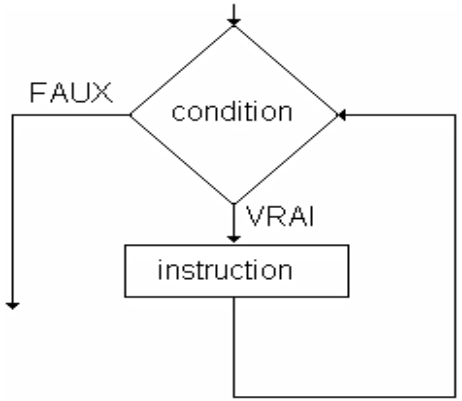
1- Alternative

Exécution des instructions en fonction d'une condition.



2- Répétitive

Exécution d'instructions plusieurs fois en fonction d'une condition.



Structure alternative

- Notamment appelée structure conditionnelle, elle permet de créer des branchements
- Conditionner l'exécution de certaines instructions
- **L'instruction IF :**
  - Syntaxe :

```
IF Boolean_expression
        { sql_statement | statement_block }
[ ELSE { sql_statement | statement_block } ]
```

  - Ou encore :

```
IF Boolean_expression
        { sql_statement | statement_block }
[ ELSE IF Boolean_expression
        { sql_statement | statement_block } ]
[ ELSE
        { sql_statement | statement_block } ]
```

Une **seule instruction permise,** sinon utiliser le bloc **BEGIN/ END**

Exemple: Instruction IF..ELSE

```
DECLARE
@vsalaire MONEY;

BEGIN
SELECT @vsalaire =AVG(SALAIRE) FROM Employes WHERE codeDep ='inf';

          IF (@vsalaire>6000) UPDATE Employes SET Salaire =
          Salaire + 0.01*salaire WHERE codeDep ='inf';

          ELSE UPDATE Employes SET Salaire = Salaire + 0.02*salaire
          WHERE codeDep ='inf';
END ;
```

Le IF..ELSE peut comporter **plus d'une instruction**

1. On **déclare** une variable @vsalaire de type MONEY
2. Le bloc d'instructions SQL **commence** par BEGIN
3. On **affecte** à la variable @vsalaire: le salaire moyen des employés du département 'inf' par une requête SELECT
4. On **vérifie** **si** le salaire moyen des employés du département 'inf' est plus élevé que 6000, **alors** on augmente le salaire des employés de ce département de 1%, **sinon** on l'augmente de 2%
5. On **termine** le bloc d'instructions par END

Structure alternative

- **L'instruction CASE :**
  - Permet d'affecter, selon une condition, une valeur à un champ dans une requête Select
  - Syntaxe :

```
CASE input_expression
        WHEN when_expression THEN result_expression [ ...n ]
        [ ELSE else_result_expression ]
END
```

  - Ou encore :

```
CASE
        WHEN Boolean_expression THEN result_expression [ ..n ]
        [ ELSE else_result_expression ]
END
```

Exemple: Instruction CASE (pour un SELECT)

```
SELECT nom, prenom, codeDep =
          CASE codeDep
               WHEN 'inf' THEN 'Informatique'
               WHEN 'rsh' THEN 'Ressources humaines'
               WHEN 'ges' THEN 'Gestion'
               WHEN 'rec' THEN 'Recherche et developpement'
               ELSE 'aucun département connu'
          END, salaire
FROM Employes ;
```

Dans cet exemple :

- On **affiche** le nom des départements **selon** (CASE) leur code (codeDep)
- On **termine** le bloc d'instructions par END

Pour un simple affichage, un **CASE est suffisant**. Toutefois une jointure avec la table Departement est beaucoup *moins coûteuse* qu'un CASE pour ce cas.

OU bien, autrement :

```
SELECT nom, prenom, codeDep,
          CASE codeDep
               WHEN 'inf' THEN 'Informatique'
               WHEN 'rsh' THEN 'Ressources humaines'
               WHEN 'ges' THEN 'Gestion'
               WHEN 'rec' THEN 'Recherche et developpement'
               ELSE 'aucun département connu'
          END AS DESCRIPTION, salaire
FROM Employes ;
```

Exemple: Instruction CASE (pour un UPDATE)

```
UPDATE Employes SET salaire =
      (CASE
         WHEN (salaire < 2500 ) THEN salaire + 0.05*salaire
         ELSE (salaire + 0.01*salaire)
       END
) ;
```

La mise à jour du salaire dépend de sa valeur initiale. Dans ce cas un **CASE est approprié**.

Dans cet exemple :

- On procède à la **mise à jour** du salaire selon sa valeur initiale.
- On **vérifie** si le salaire des employés est inférieur à 2500, **alors** on l'augmente de 5%, **sinon** on l'augmente de 1%.
- On **termine** le bloc d'instructions par END.

Structure répétitive

- Permet l'exécution d'instructions plusieurs fois en fonction d'une condition donnée.

- La répétitive est implémentée à l'aide de la **boucle WHILE**.
  - Syntaxe générale :

```
WHILE Boolean_expression
        { sql_statement | statement_block | BREAK | CONTINUE }
```

  - 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)

Une **seule instruction permise,** sinon utiliser **BEGIN/ END**

Exemple: Boucle WHILE

```
BEGIN
    WHILE (SELECT AVG (salaire) FROM Employes )<= 15000
    BEGIN UPDATE Employes SET salaire = salaire +1000;
        IF(SELECT MAX(salaire) FROM Employes) >50000
   BREAK;
        ELSE CONTINUE;
    END;
END;
```

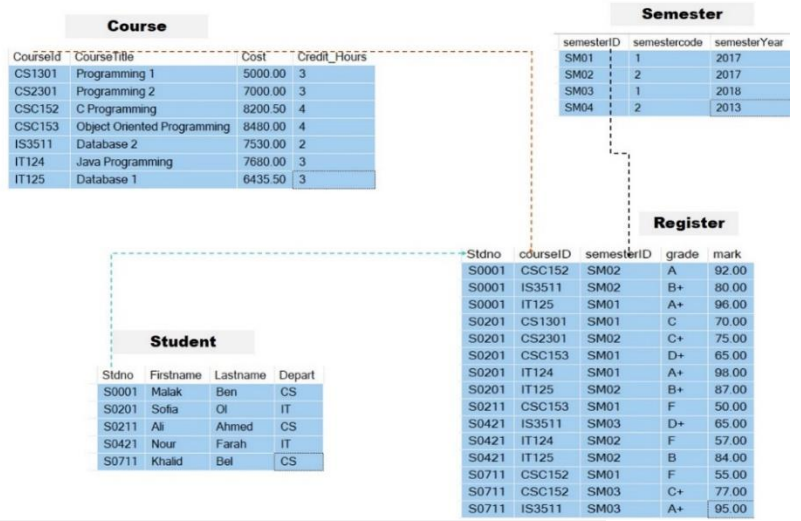
Dans cet exemple :

- On procède à la **mise à jour** du salaire des employés
- **Tant que** la moyenne est inférieure à 15 000, on **augmente** le salaire
- Mais **si** le maximum des salaires dépasse 50 000, on **arrête** (On sort de la boucle).
- **Sinon**, on **continue** à exécuter la boucle (Tant que la condition de la boucle while est satisfaite et le maximum des salaires ne dépasse pas 50000)

La mise à jour est exécutée tant que la condition de la boucle est vérifiée, sinon l'arrêt de cette dernière est forcé par le **mot Break**.

TP : Initiation T-SQL, Structures Alternatives/Répétitives

Application 1 : Initiation T-SQL



Course

CourseId	CourseTitle	Cost	Credit_Hours
CS1301	Programming 1	5000.00	3
CS2301	Programming 2	7000.00	3
CSC152	C Programming	8200.50	4
CSC153	Object Oriented Programming	8480.00	4
IS3511	Database 2	7530.00	2
IT124	Java Programming	7680.00	3
IT125	Database 1	6435.50	3

Semester

semesterID	semestercode	semesterYear
SM01	1	2017
SM02	2	2017
SM03	1	2018
SM04	2	2013

Register

Stdno	courseID	semesterID	grade	mark
S0001	CSC152	SM02	A	92.00
S0001	IS3511	SM02	B+	80.00
S0001	IT125	SM01	A+	96.00
S0201	CS1301	SM01	C	70.00
S0201	CS2301	SM02	C+	75.00
S0201	CSC153	SM01	D+	65.00
S0201	IT124	SM01	A+	98.00
S0201	IT125	SM02	B+	87.00
S0211	CSC153	SM01	F	50.00
S0421	IS3511	SM03	D+	65.00
S0421	IT124	SM02	F	57.00
S0421	IT125	SM02	B	84.00
S0711	CSC152	SM01	F	55.00
S0711	CSC152	SM03	C+	77.00
S0711	IS3511	SM03	A+	95.00

Student

Stdno	Firstname	Lastname	Depart
S0001	Malak	Ben	CS
S0201	Sofia	Ol	IT
S0211	Ali	Ahmed	CS
S0421	Nour	Farah	IT
S0711	Khalid	Bel	CS

1) Déclarez et affichez une variable.

2) Soit la base de données suivante '**CollegeDB**' :

a. Initialisez une variable à la **note maximale** des étudiants. Affichez cette variable.

b. **Si** l'étudiant appartient au département de 'CS', **alors** afficher le Nom, le Département, et les Cours qu'il suit; **Sinon**, afficher que l'étudiant n'appartient pas au département 'CS'

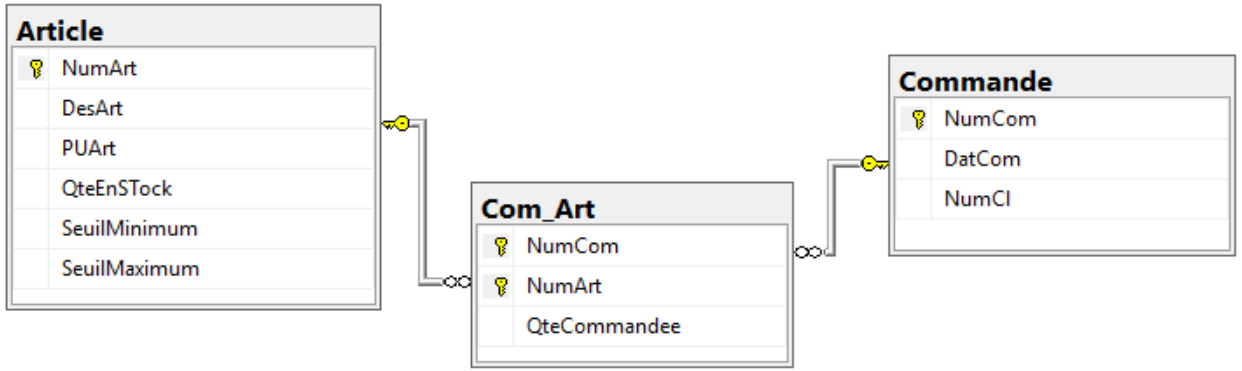
(Faire des tests pour les étudiants 'S0421' et 'S0001')

c. Afficher le *semestreID*, *semesterYear*, *courseID*, *Stdno*, *Firstname* des étudiants ayant un cours dans **chaque année du semestre** sans utiliser la clause Group by.

Application 2 : Structures Alternatives/Répétitives

Soit la base de données suivante '**GestionC**' :

Voir Fichier :
Application 2 T-SQL



1) Ecrire un programme qui **vérifie 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 » **sinon** « Stock disponible ».

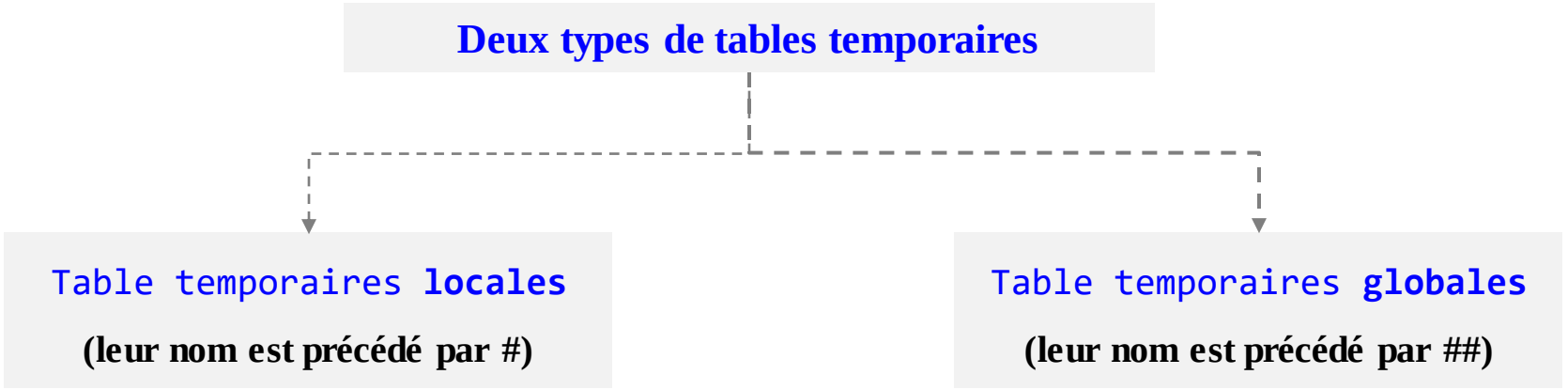
2) Ecrire un programme qui affiche la **liste des articles** (Numéro, Désignation et Prix) avec **en plus une colonne 'Observation'** qui affiche : « Non Disponible » **si** la quantité en stock est égale à 0, « Disponible » **si** la quantité en stock est supérieure au stock Minimum et « à Commander » **sinon**.

3) Ecrire un programme qui **augmente les prix** de 10% **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, **et affiche 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.

4) 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 de la commande <=10.000 DH) ou d'une « **commande spéciale »** (montant de la commande > 10.000 DH).

Tables temporaires

- Les **tables temporaires** permettent de stocker, manipuler et traiter un ensemble de données temporaires
- Toutes les fonctions SQL s'utilisent sur cette table temporaire
- La table est stockée dans la base système **TempDB** et existe jusqu'au **redémarrage du serveur** (Elles sont automatiquement supprimées dès que la connexion en cours est terminée)**.**



Création tables temporaires

Deux façons pour créer les tables temporaires

1ère Façon: L'approche CREATE TABLE

```
CREATE TABLE #name_of_temp_table (
        column_1 datatype,
        column_2 datatype,
        column_3 datatype,
        .......
        column_n datatype
)
```

2ième Façon: L'approche SELECT INTO

```
SELECT column_1, column_2, column_3,...
INTO #name_of_temp_table
FROM Table_name
WHERE condition
```

Tables temporaires

1- L'approche CREATE TABLE

- **Exemple :**

```
Create Table #T1 (ID int Primary Key, Prenom varchar(15))
Insert into #T1 values (1, 'Paul')
Select * from #T1
```

- **Ou encore :** À partir d'une table existante créant ainsi une table « copie » de la table source avec exactement les mêmes noms de colonne et types de données

```
Create Table #Table (NumArt int, PUArt int)
Insert into #Table
Select NumArt, PUArt From Article
```

Tables temporaires

- **2- L'approche SELECT INTO**
  - **Exemple :**

```
SELECT NumArt, DesArt, PUArt
INTO #Table_Temp FROM Article
WHERE PUArt > 300
```

Table temporaire créée automatiquement

  - **Ou encore :** Pour une table globale

```
select * into ##Table_Globale from Article
```

Table Temporaire

Table Source

Exemple: Création de tables temporaires

- Ecrire 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)
  - Le programme est le suivant :

```
Create Table #T1 (NumCom int, DatCom Date, MontantCom float)
Insert into #T1
Select Top 5 C.NumCom, DatCom, Sum(PUArt*QteCommandee) as Mt From Commande C
          inner join Com_Art CA on C.NumCom=CA.NumCom
          inner join Article A on A.NumArt=CA.NumArt
          Group by C.NumCom, DatCom Order by Mt Desc
```

À retenir

- Dans Transact-SQL, la **déclaration** de variable se fait par le mot **DECLARE**
- Les variables locales sont précédées du **symbole @**
- L'**affectation de valeurs** aux variables se fait par **= ou le SET**
- L'**affectation** de valeurs **issues de la base de données** à une variable se fait par un **SELECT**
- Chaque **bloc d'instructions** commence par **BEGIN** et fini par **END**
- Si une instruction **SELECT** apparaît dans une 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