

MICROSOFT SQL SERVER

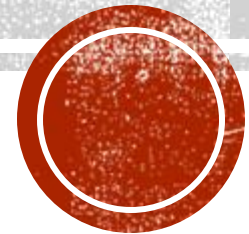


Prof. Maria EL HAIBA

Docteur en Informatique



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



4

INTRODUCTION AU LANGAGE T-SQL (PART 3)

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



Procédures stockées

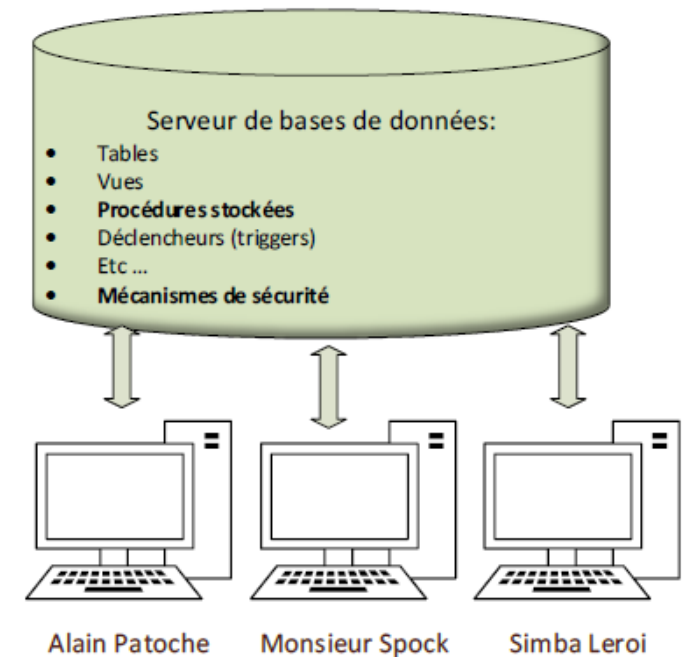
SECTION 1



Procédure Stockée

➤ Définition

- Une procédure stockée est un ensemble d'instructions SQL **précompilées** stockées dans le **serveur** de bases de données.
- C'est en fait un **objet de la Base de Données** comme les tables, ou les vues,...
- Les procédures stockées sont une suite de programmes créés et exécutés du côté serveur, destinées à être **appelées** à tout moment par **un ou plusieurs clients** de la base de données.



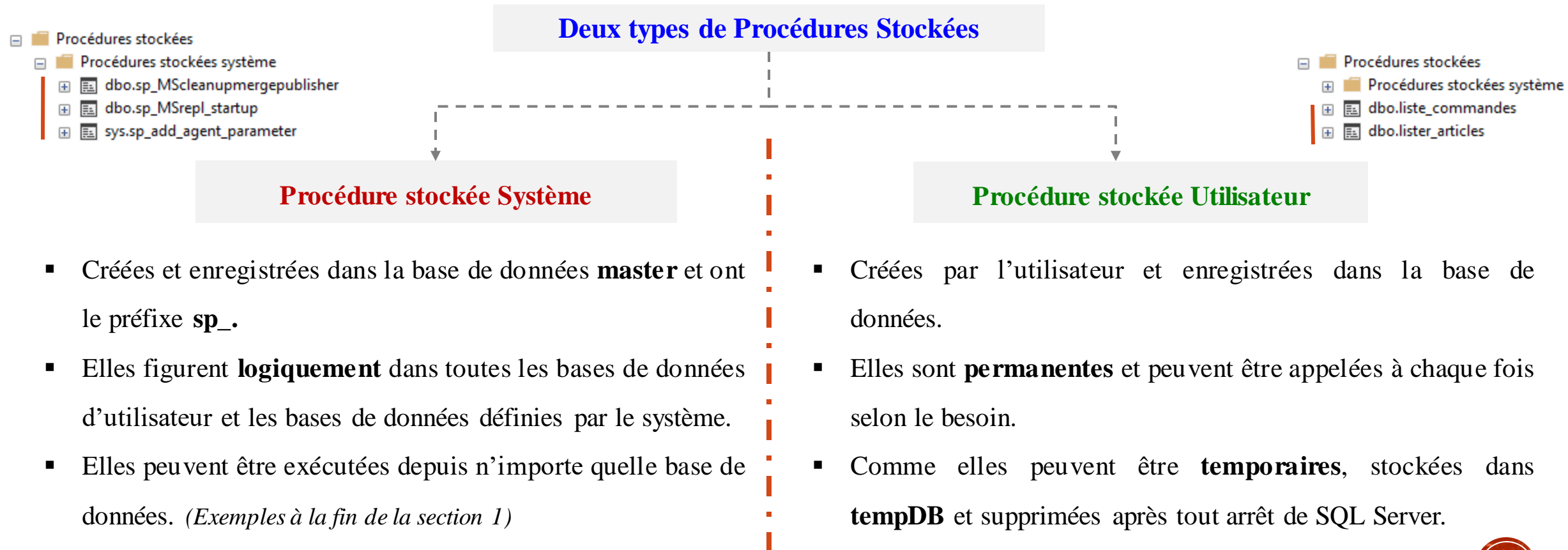
Procédure Stockée : Avantages

➤ Avantages d'utilisation d'une Procédure stockée

- **Rapidité d'exécution**, puisque les procédures stockées sont déjà compilées.
- **Clarté du code** : En particulier lorsque l'instruction SQL est longue et complexe. Il vaut mieux utiliser l'appel d'une procédure que l'instruction SQL.
- **Réutilisation** de la procédure stockée avec Possibilité d'exécuter un **ensemble de requêtes SQL**.
- **Modularité** dans le sens où elle facilite le travail d'équipe. Plusieurs clients / utilisateurs peuvent travailler sur la même BD en utilisant les procédures stockées.
- **Prévention d'injections SQL**, Tout ce que nous voyons est le nom de la procédure (et ses paramètres). Les objets que la procédure manipule ne sont pas visibles pour les autres.



Procédure Stockée : Types



Création d'une procédure

Syntaxe simplifiée de **Création d'une procédure stockée** avec T-SQL :

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }  
[ schema_name. ] procedure_name  
[ { @parameter data_type }  
[ OUT | OUTPUT ]  
AS  
{ [ BEGIN ] sql_statement [ ; ] [ ...n ] [ END ] }  
[ ; ]
```



Toute procédure peut être exécutée par appel de son nom
avec ou sans paramètres.

Explication

- 1- CREATE PROCEDURE : indique que l'on veut créer une procédure stockée.
OR ALTER est **optionnel**, indique que l'on veut modifier la procédure stockée si celle-ci existe déjà.
- 2- @parameter data_type : On doit fournir la liste des paramètres de la procédure avec le type de données correspondant à chacun des paramètres. A noter que les paramètres sont **précédées du symbole @**.
- 3- [OUT | OUTPUT] : Indique la direction en OUT ou en OUTPUT des paramètres de la procédure. Par défaut les paramètres sont en IN.
- 4- AS : mot réservé qui annonce le début du corps de la procédure et la fin de la déclaration des paramètres
- 5- BEGIN
Bloc SQL ou Transact-SQL
END;



Procédure Stockée - Sans paramètres

- La procédure stockée exécute un traitement donné qui ne dépend d'aucune valeur provenant de l'application appelante.
- Exemple d'une procédure (**sans paramètres**) pour **afficher la liste des commandes**.

```
USE GestionC;  
GO  
CREATE PROCEDURE liste_commandes  
AS  
BEGIN  
SELECT * FROM Commande  
/*SELECT...*/  
END;
```

Exécution d'une procédure

Pour **exécuter** une procédure stockée, on utilise une des alternatives suivantes:

- Procedure_name
- **EXEC** Procedure_name
- **EXECUTE** Procedure_name

```
EXECUTE liste_commandes;
```

Résultat :

	NumCom	DatCom	NumCl
1	1	2020-03-12	1
2	2	2020-05-13	2
3	3	2020-08-15	3

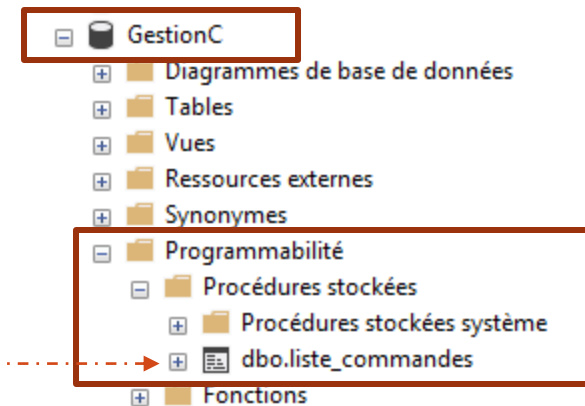


Procédure Stockée - Emplacement

- Après création, vos procédures stockées se trouvent au niveau du SSMS à l'onglet :

Programmabilité > **Procédures stockées** de la base de données en question

```
USE GestionC;  
GO  
CREATE PROCEDURE liste_commandes  
AS  
BEGIN  
SELECT * FROM Commande  
/*SELECT...*/  
END;
```



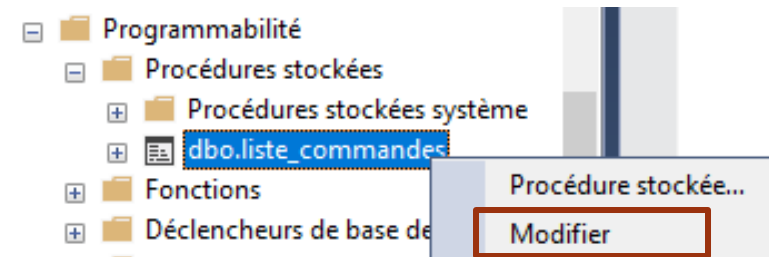
Procédure Stockée - Modification/Suppression

- Pour **MODIFIER** une procédure stockée, on utilise l'instruction suivante :

- `ALTER PROCEDURE nom_procedure`

```
ALTER PROCEDURE liste_commandes
AS
BEGIN
/*Fonctionne de la même manière que le CREATE*/
END;
```

Ou bien



Faire **un clic droit** sur la procédure puis **Modifier**.

Vous aurez le **code** de la procédure en **ALTER**

- Pour **SUPPRIMER** une procédure stockée, on utilise :

- `DROP PROCEDURE nom_procedure`

```
DROP PROCEDURE nom_procedure
```

Ou bien

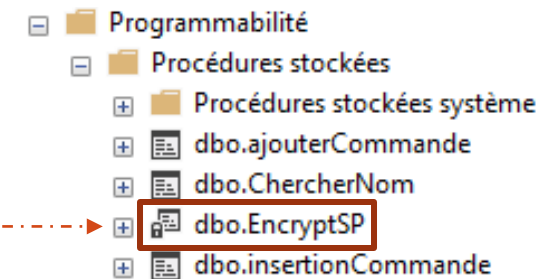
De la même manière que la modification à partir de l'explorateur d'objets du SSMS, vous pouvez faire **un clic droit** sur la procédure puis **Supprimer**.



Procédure Stockée - Cryptage

- Il est possible de rendre le **code** de la procédure stockée **inaccessible**, il suffit pour cela de procéder à un **cryptage**.
- La procédure stockée pourra être exécutée par un programme externe mais son contenu sera **illisible** que cela soit pour son propriétaire ou pour d'autres utilisateurs.

```
CREATE OR ALTER PROCEDURE EncryptSP  
[WITH ENCRYPTION] – Mot clé optionnel si vous voulez crypter le  
code de votre procédure (Empêcher de le rendre visible aux autres)  
AS  
BEGIN  
/*Bloc d'instructions*/  
END;
```



Procédure Stockée - Avec des paramètres en entrée

- Les paramètres sont par **défaut en IN**. L'indiquer provoque une **erreur**.
- **Exemple 1** d'une procédure (**avec paramètres**) pour l'insertion de nouvelles commandes.

```
CREATE PROCEDURE insertionCommande  
  (@pnum int, @pdate date, @pnumcl int)  
AS  
BEGIN  
  INSERT INTO Commande(NumCom, DatCom, NumCl)  
  VALUES(@pnum, @pdate ,@pnumcl)  
END;
```

Exécution

- Il ne faut pas oublier de **passer les paramètres** de la procédure lors de l'exécution.
- Les paramètres doivent être **précédées par le symbole @**.
- Dans ce cas, Il faudra **fournir la valeur des paramètres**.

```
EXECUTE insertionCommande  
  @pnum = 6,  
  @pdate = '12-03-2023',  
  @pnumcl = 5;
```



Exemple 1 - Suite

- A noter que MS SQL Server peut accepter la passation des paramètres **dans n'importe quel ordre**.
- Toutefois, il est **conseillé** de passer les paramètres dans **l'ordre de leur apparition** dans la procédure.
- On aurait très bien pu faire ceci , le paramètre @pnum est fourni en dernier :

```
EXECUTE insertionCommande  
@pdate = '12-03-2023',  
@pnumcl = 5,  
@pnum = 6;
```

Exécution sans erreur. Cependant, Il est très recommandé de garder le même ordre. Son utilité apparaît au moment du Débogage d'une procédure qui ne marche pas.



- OU bien, passer directement les valeurs des paramètres :

```
EXECUTE insertionCommande  
'12-03-2023', 5, 6;
```

Exécution avec erreur. Dans ce cas, il est nécessaire de garder le même ordre d'apparition dans la procédure.



Procédure Stockée - Quelques exemples

Exemple 2: Sélection des articles d'une commande particulière (dont le numéro est donné en paramètre).

```
/*Création Procédure*/  
CREATE PROCEDURE lister_articles (@pnumC INT)  
AS  
BEGIN  
SELECT NumArt, QteCommandee FROM Com_Art  
WHERE NumCom = @pnumC;  
END;  
/*Exécution*/  
EXECUTE lister_articles  
@pnumC=1; -- Ou bien EXEC lister_articles 1
```

→ **Résultat**

	NumArt	QteCommandee
1	1	10
2	2	5
3	3	4

Exemple 3: d'une procédure pour la sélection des clients en utilisant le **LIKE**.

```
/*Création Procédure*/  
CREATE PROCEDURE ChercherNom (@pnom VARCHAR (50))  
AS  
BEGIN  
SELECT * FROM Client WHERE Nom Like '%' + @pnom + '%';  
END;  
/*Exécution*/  
EXECUTE ChercherNom  
@pnom='sa'
```

→ **Résultat**

NumCl	Nom	Prenom	Ville
2	Oussama	Fihri	Salé



Application 1 : Procédures Stockées

En gardant toujours la même base de données '**GestionC**' :

- 1) Créer une procédure stockée nommée **Liste_Articles** qui affiche la liste des **articles** en mentionnant le **numéro** et la **désignation** pour chaque article.
- 2) Créer une procédure stockée nommée **NbrArt_ParCom** qui affiche le **numéro** de la commande, sa **date** et **calcule** le **nombre d'articles** par commande.
- 3) Créer une procédure stockée nommée **Com_Periode** qui affiche la liste des **commandes effectuées** entre **deux dates** données en paramètre (*Exemple : Commandes effectuées entre le 10/1/2020 et le 10/1/2021*).
- 4) Créer une procédure stockée nommée **TypeCom_Periode** qui affiche la liste des commandes effectuées entre deux dates passées en paramètres. (A noter qu'il est possible d'exploiter la procédure précédente)
 - En plus si le nombre de ces commandes est supérieur à 100, afficher « Période rouge ».
 - Si le nombre de ces commandes est entre 50 et 100, afficher « Période jaune »
 - Sinon afficher « Période blanche »

Procédure Stockée - Avec des paramètres en sortie

- Une procédure stockée avec des paramètres en sortie est une procédure **qui retourne un ou plusieurs paramètres OUT**, et qui sont utilisés pour retourner des données à l'application appelante.
- Les paramètres en sortie sont **spécifiés** par OUT | OUTPUT .
- **Exemple 1** d'une procédure (**avec paramètres**) en **OUTPUT** qui renvoie le **nombre de commandes**

```
Create Procedure Nbr_Commandes (@Nbr int output) as
Set @Nbr = (Select count(NumCom) from Commande)

--Exécuter cette procédure pour afficher le nombre de commandes
Declare @n int
Exec Nbr_Commandes @n Output
Print 'Le nombre de commandes est : ' + convert(varchar,@n)
```

Pour appeler une procédure stockée avec des paramètres de sortie, procédez comme suit:

- Premièrement, déclarez des variables pour contenir les valeurs renvoyées par les paramètres de sortie
- Après, utilisez ces variables lors de l'appel de la procédure stockée.
- Il ne faut pas oublier de **spécifier le mot OUTPUT** lors de l'exécution.

Procédure Stockée - Exemple 2

- Les procédures utilisant des paramètres de sortie peuvent avoir ou ne pas avoir (selon le besoin) des **paramètres en entrée**.
- **Exemple 2** d'une procédure (**avec paramètres**) en **OUTPUT** pour **afficher le nom des clients**.

```
Create Procedure ChercherNom2  
(@pnum int, @pnom varchar(50) out)  
AS  
begin  
select @pnom = Nom  
from Client where NumCl =@pnum;  
end;
```

Exécution

```
declare @pnum int = 2 ;  
declare @pnom varchar(50);  
execute ChercherNom2  
@pnum ,  
@pnom output;  
print @pnom; --select @pnom as 'Nom'
```



Procédure Stockée - Avec Valeur de retour

- Par défaut, lorsque nous exécutons une procédure stockée dans SQL Server, elle renvoie **une valeur entière** qui indique l'état d'exécution de la procédure. La valeur 0 indique que la procédure s'est terminée avec succès, et les valeurs non nulles indiquent une erreur.
- Nous pouvons **changer la valeur de retour** dans le code de la procédure en utilisant la commande **RETURN**.

```
Create Procedure NbrCom @desa varchar(50)
as
begin
    ► return (select count(NumCom)
              from Com_Art CA inner join Article A
              on CA.NumArt=A.NumArt where DesArt=@desa);
end
```

Limite :

- La procédure ne peut retourner qu'une **SEULE** valeur qui est de **type INT**.

Exécution

```
declare @c int;
exec @c =NbrCom 'Jus'
select @c
```



Procédure Stockée - Paramètres de **sortie** VS Valeur de **retour**

Valeur de retour :

- Il est préférable d'utiliser une valeur de retour lorsque vous souhaitez renvoyer **un SEUL élément avec uniquement un type de données ENTIER**,
- Généralement, la valeur de retour sert à informer le succès ou l'échec de la procédure stockée.

Paramètre de sortie :

- Il est préférable d'utiliser les paramètres de sortie lorsque vous souhaitez renvoyer **UN ou PLUSIEURS éléments** avec un type de données.



Application 2 : Procédures Stockées OUTPUT

En gardant toujours la même base de données '**GestionC**' :

- 1) Créer une procédure stockée nommée **Nbr_ArtCom** qui retourne le nombre d'articles d'une commande dont le numéro est donné en paramètre, sous la forme suivante :

Le nombre d'articles de la commande numéro ... est : ... (*Exemple: Commande N°1*)

- 2) Créer une procédure stockée nommée **Montant_C** qui reçoit le numéro de commande et affiche le montant total de cette commande.
- 3) Créer une procédure stockée nommée **Type_Montant** qui renvoie un code de retour. Si le montant d'une commande est supérieur à 10.000, la procédure renvoie 1. Si le montant d'une commande est inférieur à 10.000, la procédure renvoie 2. Si une erreur système a lieu, la procédure renvoie 3.



Procédures Stockées Système - Quelques exemples

➤ Quelques procédures système utiles pour l'utilisateur :

- **sp_help spName:** Permet de voir des informations sur la procédure stockée ; i.e. les noms de paramètres, les types de données... Cette procédure système peut également être utilisée avec n'importe quel autre objet de la base de donnée; i.e. Table, View, Trigger...
- **sp_helptext spName:** Permet de voir le texte de la procédure spName
- **sp_depends spName:** Permet de voir et afficher des informations sur les dépendances de la procédure stockée "spName". Les références à des objets en dehors de la base de données actuelle ne sont pas signalées.

Exemple:

sp_depends NbrArt_ParCom ;


	name	type	updated	selected	column
1	dbo.Commande	user table	no	yes	NumCom
2	dbo.Commande	user table	no	yes	DatCom
3	dbo.Com_Art	user table	no	yes	NumCom
4	dbo.Com_Art	user table	no	yes	NumArt

! /* Peut aussi être utilisée pour tout objet de la BD tel qu'une table (Ex.: Table Commande)*/

sp_depends Commande ;

	name	type
1	dbo.Com_Periode	stored procedure
2	dbo.insertionCommande	stored procedure
3	dbo.liste_commandes	stored procedure
4	dbo.Nbr_Commandes	stored procedure
5	dbo.NbrArt_ParCom	stored procedure
6	dbo.TypeCom_Periode	stored procedure





Fonctions Stockées / Système

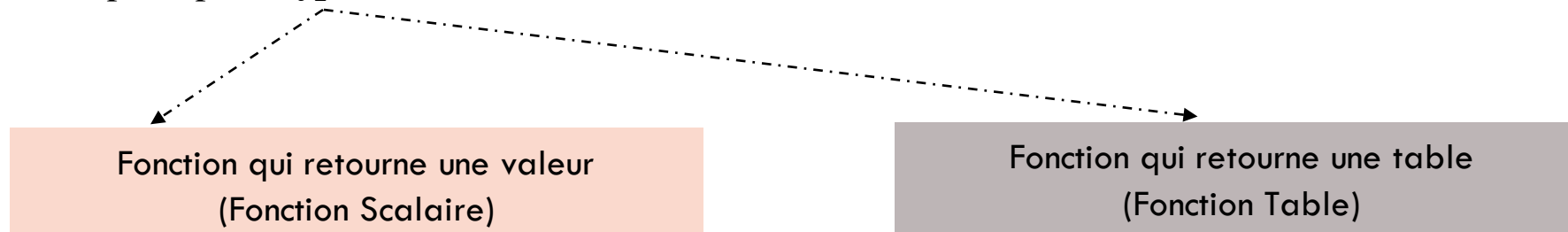
SECTION 2



Fonctions T-SQL

➤ T-SQL permet de créer des UDF (User Define Functions), notamment appelées **Fonctions Stockées**

- Dans T-SQL une fonction est considérée comme un objet
- Après sa création, elle fait partie de la BD
- Peut être utilisée dans du code T-SQL (triggers, transactions, etc.)
- Deux principaux **types** de fonctions :



➤ T-SQL possède notamment des fonctions prédéfinies dans SQL Server, appelées **Fonctions Système**



Fonction Stockée - Fonction **Scalaire**

➤ Définition d'une fonction stockée

- Les fonctions stockées sont des **procédures stockées qui retournent des valeurs**.
- Leurs définitions sont légèrement différentes d'une procédure stockée mais le principe général de définition reste le même.
- **Syntaxe de création :**

```
CREATE [ OR ALTER ] FUNCTION function_name  
( [ { @parameter_name parameter_data_type } ] )  
RETURNS return_data_type  
[ AS ]  
BEGIN  
/* sql_statement */  
RETURN scalar_expression  
END  
[ ; ]
```



Fonction Stockée - Exemples

Exemple 1 : Fonction avec paramètres

```
/*Création Fonction*/
CREATE FUNCTION compterClient(@ville varchar(50))
RETURNS INT
AS
BEGIN
DECLARE @total INT;
SELECT @total = COUNT(*) FROM Client
        WHERE ville =@ville;
RETURN @total;
END;

/*Exécution*/ --Pas besoin de la clause FROM
select dbo.compterClient('Rabat')
```

Exemple 2 : Fonction sans paramètres

```
/*Création Fonction*/
CREATE FUNCTION compterClient() RETURNS INT
AS
BEGIN
DECLARE @total INT;
SELECT @total = COUNT(*) FROM Client;
RETURN @total;
END;

/*Exécution*/
select dbo.compterClient()
```



Pour exécuter une fonction qui retourne une valeur, il faudra :

- Utiliser la commande **SELECT**, suivie du Nom de la fonction.
- EN Précisant le **schéma de la BD** (dbo.nomFonction)
- Passer les **valeurs des paramètres** pour la fonction (Si c'est le cas)

Fonction Stockée - Fonction **Table**

➤ Syntaxe de création d'une fonction table

- Il n'y a pas de **BEGIN** et **END** pour une fonction qui **retourne une table**.
- La fonction **TABLE** se comporte comme une table.
- **Syntaxe simplifiée de création :**

```
CREATE [ OR ALTER ] FUNCTION function_name  
( [ { @parameter_name parameter_data_type } ] )  
RETURNS TABLE  
[ AS ]  
RETURN [ ( select_statement ) ]  
[ ; ]
```

--Il n' y pas de **BEGIN...END**



Fonction Stockée - Exemple

Exemple

```
/*Création Fonction*/  
CREATE FUNCTION ChercherCLIENT (@ville varchar(50))  
RETURNS TABLE  
AS  
RETURN(  
SELECT nom, prenom, ville  
FROM Client  
WHERE @ville =ville  
) ;  
  
/*Exécution*/  
select * from ChercherCLIENT ('Florida')
```



Pour **exécuter une fonction qui retourne une table**, il faudra :

- Utiliser la commande **SELECT**, avec la clause **FROM**.
- Déclarer les **paramètres** et leur **affecter des valeurs** si la fonction a des paramètres en IN.

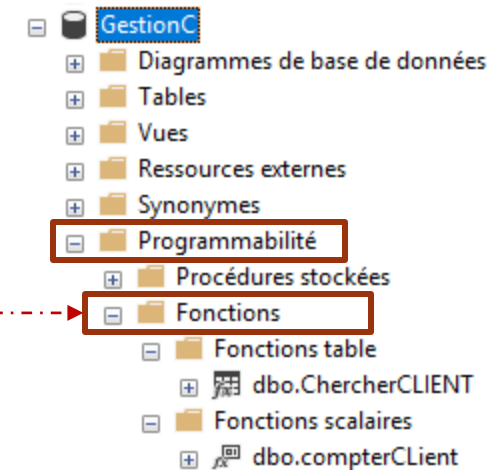


Fonction Stockée - Emplacement/Modification/Suppression

- Après création, vos fonctions se trouvent au niveau du SSMS à l'onglet :

Programmabilité > **Fonctions** de la base de données en question

Emplacement après création des fonctions



- Pour **MODIFIER** une fonction stockée, on utilise :

```
ALTER FUNCTION nom_Fonction
```

- Pour **SUPPRIMER** une fonction stockée, on utilise :

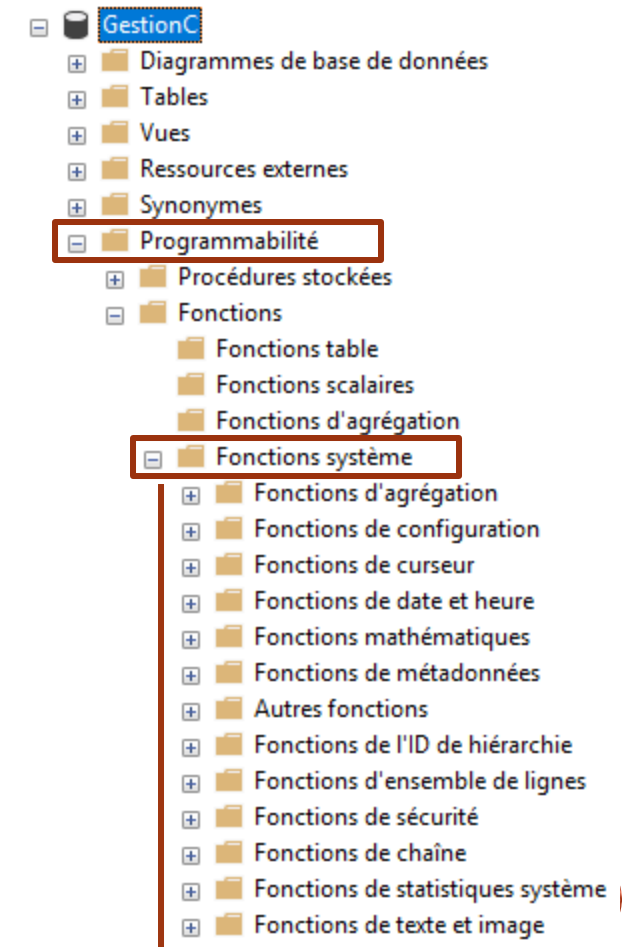
```
DROP FUNCTION nom_Fonction
```



Fonction Système

➤ Quelques fonctions systèmes utiles

- Les **fonctions systèmes** sont préfinies dans SQL Server et peuvent être exploitées par l'utilisateur à tout moment.
 - Fonctions d'**agrégation** : Sum(), Avg(), Max(), Min(), Count(), ...
 - Fonctions de **date et heure** : Getdate(), Sysdatetime(), Datediff(),...
 - Fonctions de **curseur** : @Fetch_Status, @Cursor_Rows,...
 - Fonctions de **chaîne** : Upper(), Lower(), Concat(), Replace(),...
 - Fonctions **mathématiques** : Abs(), Cos(), Exp(), Log(),...
 - **Autres Fonctions systèmes** : Cast(), @@Rowcount, @@Trancount, Isnull(),...



Fonction Système

➤ Quelques exemples d'utilisation des fonctions système

```
select GETDATE() as 'Date et heure' --Returns 2022-05-11 23:05:07.603
select SYSDATETIME()
select DATEDIFF(year, '2011/08/25', '2019/08/25') AS DateDiff; --Returns 8
----
select ABS(-10.5) --Returns 10.5
select POWER (2,3) --Returns 8
select SQUARE(9) --Returns 81
---
select RIGHT('ABCDEF',3) -- Returns DEF
---
select ville, REPLACE(ville, 'Paris', 'Madrid') as changedville from Client
---
select NumCom, DatCom, NumCl + ' - ' + Cast(NumCl as nvarchar) as Name_ID from Commande
select cast(getdate() as Date) --Returns 2022-05-11
select ISNULL(NULL,10); -- Returns 10
---
select @@TRANCOUNT --Returns 0
select DB_NAME() --Returns GestionC
select @@ERROR --Returns 0
```



Procédure Stockée VS Fonction Stockée

Procédure Stockée :



- La procédure ne retourne pas de valeur sauf pour le type entier (*Cas de procédure avec Valeur de retour*).
- On peut jouer avec les tables, faire appel à des procédures et des fonctions; et notamment contenir des transactions.
- On peut aussi y mettre des variables « OUTPUT » pour retourner des valeurs même non entières.
- Ne permet pas de retourner des tables (*Sinon, il faudra passer toutes les colonnes en paramètres*).

Fonction Stockée :



- Une fonction retourne une valeur scalaire ou carrément une table.
- On ne peut pas appeler une procédure stockée ou faire du LMD sur des tables (*Actions modifiant l'état de la BD*).
- Ne peut en aucun cas contenir une transaction.
- Peut être utilisée au sein d'une requête ou d'une procédure.



TP : Procédure et Fonction Stockée



Exercice 1 : Procédure Stockée

En gardant toujours la même base de données '**GestionC**' :



Voir Série N°3: Procédures & Fonctions

Créer une procédure stockée nommée **AjouterCommande** qui permet de créer une commande pour un client. Elle reçoit un numéro de commande, un numéro de client, une date, un numéro d'article et la quantité commandée.

Cette procédure fait les opérations suivantes :

- 1) Si l'**article n'existe pas** ou si la **quantité demandée n'est pas disponible**, afficher un **message d'erreur**
- 2) Sinon, Si la **commande** introduite en paramètre **n'existe pas**, la **créer** (*à insérer dans la table Commande*)
 - **Ajouter** ensuite la ligne de commande (*à insérer dans la table Com_Art*)
 - **Mettre à jour** la quantité en stock après achat (*dans la table Article*)
 - **Mettre à jour** le montant de l'achat pour l'article (*dans la table Com_Art*)



Exercice 2 : Fonction Stockée

En gardant toujours la même base de données '**GestionC**' :

Ecrire un programme qui permet de retourner le détail de **chaque** commande (Afficher la désignation, le prix, la quantité commandée et le montant par ligne) en utilisant **les fonctions** (Pour afficher le détail) et **les curseurs** (Pour parcourir les commandes).

Le résultat devrait apparaître comme suit :

Résultats			
Le détail de la commande 1 est:			
DesArt	PUArt	QteCommandee	MtArt

Biscuit	12	10	120
Jus	20	5	100
Yogurt	5	4	20
(3 lignes affectées)			
Le détail de la commande 2 est:			
DesArt	PUArt	QteCommandee	MtArt

Ordinateur	7000	2	14000

