Module08 – Procédures, fonctions et curseurs

Objectifs

- Procédures
- Fonctions
- Curseurs

Procédure – Définition

- Une procédure SQL est un ensemble d'instructions, ici T-SQL, qui est stockée dans le serveur de bases de données
- Elle permet d'exécuter des tâches plus complexes et proches du domaine d'affaire comme dans les langages de programmation classiques
- Elle peut prendre des paramètres en entrée, en sortie et « renvoyer » des résultats (result set)

https://learn.microsoft.com/fr-fr/sql/relational-databases/stored-procedures/stored-procedures-database-engine

Procédure – Création – Syntaxe

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter_name [ type_schema_name. ] data_type }
        [ VARYING ] [ NULL ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

- Ne pas utiliser « ; **number** » : c'est déprécié
- **VARYING** : utilisé pour les curseurs seulement
- **OUT** : peut être utilisé pour renvoyer une valeur
- **READONLY** : paramètre en lecture seule
- **RECOMPILE** : ne pas réutiliser le plan d'exécution
- **EXECUTE AS** : change le contexte d'exécution (Sécurité : compte utilisateur)

https://learn.microsoft.com/fr-fr/sql/t-sql/statements/create-procedure-transact-sql
https://learn.microsoft.com/fr-fr/sql/t-sql/statements/execute-as-clause-transact-sql

Procédure – Création et appel – Exemple 1

```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SET NOCOUNT ON;
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment;
GO
```

- NOCOUNT = ne pas compter les résultats des requêtes
- Renvoie un ensemble de données

```
EXECUTE HumanResources.uspGetAllEmployees;
GO
-- Or
EXEC HumanResources.uspGetAllEmployees;
GO
-- Or, if this procedure is the first statement within a batch:
HumanResources.uspGetAllEmployees;
```

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql

Procédure – Création et appel – Exemple 2

```
CREATE PROCEDURE dbo.uspMultipleResults
AS
SELECT TOP(10) BusinessEntityID, Lastname, FirstName FROM Person.Person;
SELECT TOP(10) CustomerID, AccountNumber FROM Sales.Customer;
GO
```

Une procédure peut renvoyer plusieurs ensemble de données

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql

Procédure – Création et appel – Exemple 3

```
IF OBJECT_ID ( 'Production.uspGetList', 'P' ) IS NOT NULL
    DROP PROCEDURE Production.uspGetList;
GO
CREATE PROCEDURE Production.uspGetList @Product VARCHAR(40)
    , @MaxPrice MONEY
    , @ComparePrice MONEY OUTPUT
    , @ListPrice MONEY OUT
AS
    SET NOCOUNT ON;
    SELECT p.[Name] AS Product, p.ListPrice AS 'List Price'
    FROM Production.Product AS p
    JOIN Production.ProductSubcategory AS s
      ON p.ProductSubcategoryID = s.ProductSubcategoryID
    WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;
-- Populate the output variable @ListPprice.
SET @ListPrice = (SELECT MAX(p.ListPrice)
    FROM Production.Product AS p
    JOIN Production.ProductSubcategory AS s
      ON p.ProductSubcategoryID = s.ProductSubcategoryID
    WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice);
-- Populate the output variable @compareprice.
SET @ComparePrice = @MaxPrice;
GO
```

- 4 paramètres ici dont deux en sortie :
  - **@Product** : permet de filtrer les catégories des produits considérés
  - **@MaxPrice** : prix maximum des produits considérés
  - **@ComparePrice** : prix de comparaison renvoyé
  - **@ListPrice** : maximum des prix vente recommandé des produit de l'ensemble
- La première requête va être renvoyée comme ensemble de données : ensemble des prix de vente conseillés inférieurs à un prix maximum pour les produits de la catégorie considérée
- La variable @ListPrice sera mise à jour avec la valeur renvoyée par le SELECT (donc une ligne / une colonne pour que cela soit possible)
- La variable @ComparePrice est affectée à @MaxPrice

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql

Procédure – Création et appel – Exemple 3

```
DECLARE @ComparePrice MONEY, @Cost MONEY;
EXECUTE Production.uspGetList '%Bikes%', 700,
    @ComparePrice OUT,
    @Cost OUTPUT
IF @Cost <= @ComparePrice
BEGIN
    PRINT 'These products can be purchased for less than
    $'+RTRIM(CAST(@ComparePrice AS VARCHAR(20)))+'.'
END
ELSE
    PRINT 'The prices for all products in this category exceed
    $'+ RTRIM(CAST(@ComparePrice AS VARCHAR(20)))+'.';
```

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql

Fonction – Définition

- Les fonctions sont similaires aux procédures mais renvoie un résultat similaire à ce que font les fonctions aux sens programmation impérative :
  - Renvoie une valeur seule : fonction scalaire
  - Renvoie un tableau de valeurs (table) : fonction table
- Les fonctions **ne peuvent pas renvoyer** des **types utilisateur** de type table (CREATE TYPE UT_XYZ AS TABLE (…))
- Les fonctions **ne peuvent pas utiliser** des **fonctions non déterministes** (Ex. NEWID(), RAND(), etc.)
- Exemples d'utilisations :
  - CHECK
  - T-SQL comme SELECT
  - Dans d'autres fonctions
  - …

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql

Fonction scalaire – Syntaxe

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type [ NULL
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Fonction scalaire - Exemple

```
CREATE OR ALTER FUNCTION UDF_Min(@valeur1 INT, @valeur2 INT)
RETURNS INT AS
BEGIN
	DECLARE @min INT = @valeur2;
	IF @valeur1 < @valeur2
	BEGIN
		SET @min = @valeur1;
	END

	RETURN @min;
END;

DECLARE @min INT = dbo.UDF_Min(10, -10);
SELECT @min;

SELECT dbo.UDF_Min(10, -10)
```

Fonction table – Syntaxe

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type [ NULL ]
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

Fonction table – Exemple

```
CREATE OR ALTER FUNCTION UDF_GenererTableauEntiers(@nombreElements INT)
RETURNS @res TABLE (
                        valeur INT PRIMARY KEY
                        ) AS
BEGIN
      DECLARE @iterationCourante INT = 0;

      WHILE (@iterationCourante < @nombreElements)
      BEGIN
            INSERT INTO @res
            VALUES (@iterationCourante);

            SET @iterationCourante += 1;
      END;

      RETURN;
END;
GO

SELECT * FROM UDF_GenererTableauEntiers(10);
GO
```

Fonction – Passer outre la limite du non déterminisme

```
CREATE OR ALTER VIEW VW_NEWID AS
SELECT NEWID() AS [Guid];
GO
```

```
CREATE OR ALTER FUNCTION UDF_GenererTableauGuids(@nombreElements INT)
RETURNS
	@res TABLE (
		[index] INT PRIMARY KEY,
		[guid] UNIQUEIDENTIFIER
	) AS
BEGIN
	DECLARE @iterationCourante INT = 0;

	WHILE (@iterationCourante < @nombreElements)
	BEGIN
	INSERT INTO @res
	VALUES (@iterationCourante, (SELECT * FROM VW_NEWID));

	SET @iterationCourante += 1;
	END;

	RETURN;
END;
GO

SELECT * FROM UDF_GenererTableauGuids(10);
```

Curseur – Définition

- Un curseur est un type particulier de variables qui permet de définir un itérateur sur le résultat d'une requête
- **Il est utilisé quand il n'est pas possible d'effectuer le traitement sur un ensemble de données** (i.e. : il faut réfléchir à deux fois avant de l'utiliser !)
  - Son utilisation est moins optimal que des requêtes sur des ensembles
  - Il peut provoquer des blocages de données (réservations de lignes / tables durant un traitement)
  - Il est souvent remplaçable par d'autres instructions

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/cursors-transact-sql

Curseur – Algorithme typique

- Créer un curseur et l'affecter à une variable (**DECLARE** <nomCurseur> **CURSOR FOR SELECT** …)
- L'ouvrir (**OPEN** <nomCurseur>)
- Chercher la ligne suivante (ici la première ligne) et affecter les colonnes à des variables (**FETCH NEXT FROM** <nomCurseur> **INTO** @var1, @var2, …)
- Tant que données récupérées (**WHILE** @@FETCH_STATUS = 0)
  - Traitement sur la dernière ligne obtenue
  - Chercher la ligne suivante (ici la première ligne) et affecter les colonnes à des variables (**FETCH NEXT FROM** <nomCurseur> **INTO** @var1, @var2, …)
- **Fermer le curseur** (**CLOSE** <nomCurseur>)
- **Le désallouer** (**DEALLOCATE** <nomCurseur>)

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/cursors-transact-sql

Curseur – Syntaxe

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
     FOR select_statement
     [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]
```

```
OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

```
FETCH
          [ [ NEXT | PRIOR | FIRST | LAST
                   | ABSOLUTE { n | @nvar }
                   | RELATIVE { n | @nvar }
              ]
              FROM
          ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

```
CLOSE { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

```
DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name }
```

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/cursors-transact-sql

Curseur – Exemple

```
DECLARE Employee_Cursor CURSOR FOR
SELECT EmployeeID, Title FROM AdventureWorks2012.HumanResources.Employee;
OPEN Employee_Cursor;
FETCH NEXT FROM Employee_Cursor;
WHILE @@FETCH_STATUS = 0
    BEGIN
        FETCH NEXT FROM Employee_Cursor;
    END;
CLOSE Employee_Cursor;
DEALLOCATE Employee_Cursor;
GO
```

Insérez le code ici

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/close-transact-sql

Quelques variables

- @@ROWCOUNT : nombre de lignes affectées par la dernière instruction
- @@IDENTITY : dernière valeur de type identity de la dernière insertion
- @@ERROR/ERROR_NUMBER() : code d'erreur par la dernière instruction (0 si pas d'erreur)
- ERROR_LINE() : numéro de ligne de la dernière erreur
- ERROR_MESSAGE() : message de l'erreur
- ERROR_PROCEDURE() : nom de la procédure qui a levé une erreur
- …

https://learn.microsoft.com/en-us/sql/t-sql/functions/system-functions-transact-sql