Contents

[C02 - Getting Started With The SELECT Statement 3](#_Toc419836995)

[L1 : Using the FROM and SELECT Clause (p.62) 3](#_Toc419836996)

[L2 : Working With Data Types and Built-in Function (p.69) 3](#_Toc419836997)

[C03 - Filtering and Sorting Data 5](#_Toc419836998)

[L1 : Filtering Data with Predicates (p.94) 5](#_Toc419836999)

[L2 : Sorting Data (p.106) 5](#_Toc419837000)

[L3 : Filtering Data with TOP and OFFSET-FETCH (p.116) 5](#_Toc419837001)

[C04 - Combining Sets 6](#_Toc419837002)

[L1 : Using Joins (p.133) 6](#_Toc419837003)

[L2 : Using Subqueries, Table Expressions, and the APPLY Operator (p.149) 6](#_Toc419837004)

[L3 : Using Set Operators (p.168) 6](#_Toc419837005)

[C05 - Grouping and Windowing 8](#_Toc419837006)

[L1 : Writing Grouped Queries (p.150) 8](#_Toc419837007)

[L3 : Using Window Functions (p.172) 9](#_Toc419837008)

[Querying and Managing XML Data 10](#_Toc419837009)

[Lesson 1: Returning Results As XML with FOR XML (p.222) 10](#_Toc419837010)

[C08 - Creating Tables and Enforcing Data Integrity 12](#_Toc419837011)

[L1 : Creating and Altering Tables [p.297] 12](#_Toc419837012)

[L2 : Enforcing Data Integrity [p.313] 12](#_Toc419837013)

[C09 - Designing and Creating Views, Inline Functions, and Synonyms 14](#_Toc419837014)

[L1 : Designing and Implementing Views and Inline Functions [p.332] 14](#_Toc419837015)

[L2 : Using Synonyms [p.347] 14](#_Toc419837016)

[C10 - Inserting, Updating, and Deleting Data 15](#_Toc419837017)

[L1 : Inserting Data [p.363] 15](#_Toc419837018)

[L2 : Updating Data [p.373] 15](#_Toc419837019)

[L3 : Deleting Data [p.388] 15](#_Toc419837020)

[C11 - Other Data Modification Aspects 16](#_Toc419837021)

[L1 - Using the Sequence Object and IDENTITY Column Property [p.402] 16](#_Toc419837022)

[L2 - Merging Data [p.415] 16](#_Toc419837023)

[L3 - Using the OUTPUT Option [p.393] 17](#_Toc419837024)

[C12 - Implementing Transactions, Error Handling, and Dynamic SQL [p.443] 18](#_Toc419837025)

[L1 : Managing Transactions and Concurrency [p.444] 18](#_Toc419837026)

[L2 : Implementing Error Handling [p.467] 18](#_Toc419837027)

[L3: Using Dynamic SQL [p.482] 19](#_Toc419837028)

[C13 - Designing and Implementing T-SQL Routines 20](#_Toc419837029)

[L1 : Designing and Implementing Stored Procedures [p.502] 20](#_Toc419837030)

[L2 : Implementing Triggers [p.522] 20](#_Toc419837031)

[Lesson 3: Implementing User-Defined Functions 20](#_Toc419837032)

[XML 22](#_Toc419837033)

# C02 - Getting Started With The SELECT Statement

## L1 : Using the FROM and SELECT Clause (p.62)

* Alias forms :
  + <expression> as <alias>
  + <expression> <alias>
  + <alias> = <expression>
* Irregular identifier = identifier starts with digits, space or TSQL keyword

## L2 : Working With Data Types and Built-in Function (p.69)

* Types :
  + exact numeric : INT, NUMERIC
  + character string : CHAR, VARCHAR
  + unicode character string : NCHAR, NVARCHAR
  + approximate numeric : FLOAT, REAL
  + binary : BIT, VARBINARY
  + date and time : DATE, TIME, DATETIME2, SMALLDATETIME, DATETIME, DATETIMEOFFSET
* Fonctions de conversion
  + CAST / TRY\_CAST

SELECT CAST(‘abc’ AS INT)

* + CONVERT / TRY\_CONVERT

SELECT CONVERT(‘date’, ‘20120201’, 101)

* + PARSE / TRY\_PARSE

SELECT PARSE(‘20120201’ AS DATE USING ‘en-US’)

* Table keys :
  + Identity column property
  + Sequence object
  + Nonsequential GUIDs => NEWID
  + Sequential GUIDs => NEWSEQUENTIALID
* Current date :
  + GETDATE, CURRENT\_TIMESTAMP, GETUTCDATE, SYSDATETIME, SYSUTCDATETIME, SYSDATETIMEOFFSET
* Date & Time parts :
  + DATEPART(<type part>, ‘date’)
  + DATENAME(<type part>, ‘date’) = idem que DATEPART mais retour littéral
  + EOMONTH(‘date’)
* Add & Diff
  + DATEADD(<type part>, int, datetime)
  + DATEDIFF(<type part>, datetime, datetime)
* Offset
  + SWITCHOFFSET(datetime offset, offset) == applique la différence entre les 2 offsets à l’affichage
  + TODATETIMEOFFSET(datetime, offset) == complète le datetime avec l’offset
* Concatenation
  + ‘str1’ + ‘str2’ + ‘str3’ : option CONCAT\_NULL\_YIELDS\_NULL\_INPUT renvoie NULL si une chaîne est NULL.
  + COALESCE(<string>,<string>)
  + CONCAT(<string>,<string>) : concaténation mais remplace les chaînes NULL par une chaîne vide.
* Substring extraction and Position
  + SUBSTRING(<string>,int,int)
  + LEFT/RIGHT(<string>,int)
  + CHARINDEX(<string>,<string>) : position de la première chaîne dans la seconde.
  + PATINDEX(<pattern>,<string>) : idem que CHARINDEX mais avec un pattern
* String length
  + LEN, DATALENGTH
* String alteration
  + REPLACE(<input str>,<tobereplaced>,<by>) : écrase
  + REPLICATE(<str>,<nb occurrences>) : duplique
  + STUFF(<input str>, <position>, <nbchartodelete>,<strtoinsert>) : supprime & insert
* String formatting
  + UPPER, LOWER, LTRIM, RTRIM
  + FORMAT : FORMAT(123,’00000’) ou FORMAT(123,’d5’) => ‘00123’
* CASE expression
  + CASE WHEN THEN ELSE END : simple form and searched form
  + COALESCE, ISNULL : le type du retour est dépendant du type du premier paramètre avec ISNULL.
  + NULLIF(<string1>,<string2>) : renvoi NULL si les chaînes sont identiques.
  + IIF(<predicate>,<true\_result>,<false or unknown result>)
  + CHOOSE(<pos>,<item1>,<item2>,…)

# C03 - Filtering and Sorting Data

## L1 : Filtering Data with Predicates (p.94)

* Three-valued logic : true, false, unknown (null)
* Penser à l’utilisation des index dans l’écriture des prédicats : éviter les fonctions type ISNULL, LEFT, …
* LIKE : ne pas utiliser de « joker » en première position
* Date : forme neutre dans toutes les langues = yyyymmdd
* SARG = search argument

## L2 : Sorting Data (p.106)

* ORDER BY = unique garantie d’obtenir un résultat trié (même si index cluster)
* Dans l’ordre d’exécution des éléments d’une requête, ORDER BY arrive en dernier d’où la possibilité d’utiliser les positions ordinaux (déconseillé !)
* Attention, avec l’utilisation d’un DISTINCT, les colonnes référencées dans le ORDER BY doivent être présentes dans le SELECT
* SQL Server trie les NULL en premier
* Avec order by, le résultat obtenu est appelé « cursor » - Sans order by, résultat est de type relationnel.

## L3 : Filtering Data with TOP and OFFSET-FETCH (p.116)

* TOP + ORDER BY : le tri s’exécute avant le TOP
* TOP(n)PERCENT … ORDER BY : pourcentage du nombre de lignes à afficher, arrondi à l’entier supérieur
* TOP(n) WITH TIES : affiche les éléments identiques en se basant sur les éléments du ORDER BY.
* ORDER BY … + OFFSET x [ROW/ROWS] + FETCH [NEXT/FIRST] y [ROW/ROWS] ONLY : possibilité d’ignorer des lignes (sinon mettre 0), utilisation de variables autorisée

# C04 - Combining Sets

## L1 : Using Joins (p.133)

* CROSS JOIN : produit cartésien des colonnes de 2 tables
* INNER JOIN
* LEFT/RIGHT JOIN
* FULL JOIN = LEFT + RIGHT + INNER

Penser à créer des index sur les colonnes utilisées dans les jointures.

## L2 : Using Subqueries, Table Expressions, and the APPLY Operator (p.149)

* Self-Contained Subqueries = no dependency on the outer query.
* Correlated Subqueries = inner query has a reference to a column from the table in the outer query
* Table Expressions
* Derived tables

SELECT …

FROM ( SELECT … FROM <table>) D

* Common table expressions (CTEs) : semblable aux tables dérivées mais réutilisables

WITH C AS

(

SELECT ...FROM T1

)

SELECT ...

FROM C AS C1

INNER JOIN C AS C2 ON ...;

* Views : objet enregistré en base de données, réutilisable.
* Inline table-valued functions : similaire à une vue, mais accepte des paramètres d’entrée.

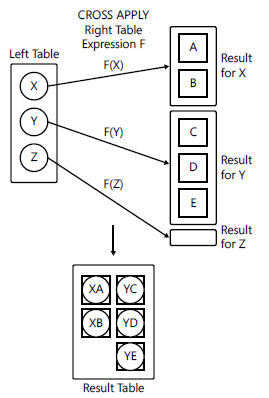
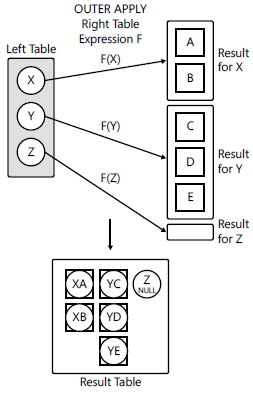
CREATE FUNCTION <function> (<@var> AS INT) RETURNS TABLE

AS

RETURN

SELECT … FROM <table>;

* CROSS APPLY – OUTER APPLY

## L3 : Using Set Operators (p.168)

* Infos valables pour les différents opérateurs :
  + Nombre de colonnes identiques dans les deux requêtes, type de donnée compatible (conversion implicited)
  + 2 NULLS sont considérés comme égaux (!),
  + le ORDER BY agit sur l’ensemble du résultat et non sur une partie de la requête,
  + le nom des colonnes du résultat est déterminé par la 1ère requête.
* UNION vs UNION ALL : le premier réalise un distinct
* INTERSECT : retourne les éléments communs aux deux ensembles, réalise un distinct
* EXCEPT : retourne les éléments qui sont uniquement présents dans l’ensemble de gauche, réalise un distinct
* Priorités d’exécution : INTERSECT > EXCEPT = UNION

# C05 - Grouping and Windowing

## L1 : Writing Grouped Queries (p.150)

* Fonctions d’aggrégation : COUNT, SUM, AVG, MIN, MAX + GROUP BY
* De manière générale, les fonctions ignorent les NULL
* Différence entre COUNT(\*) et COUNT([colonne]) : le second ignore les NULL
* Possibilité d’ajouter un DISTINCT dans les fonctions : COUNT(DISTINCT [colonne])
* GROUPING SET

SELECT [colonne]

FROM [table]

GROUP BY GROUPING SETS

(

([colonne A], [colonne B]),

([colonne B])

)

* GROUP BY CUBE([colonne A], [colonne B]) équivalent à :

([colonne A], [colonne B])

([colonne A])

([colonne B])

()

* GROUP BY ROLLUP([colonne A], [colonne B], [colonne C]) équivalent à :

([colonne A], [colonne B], [colonne C])

([colonne A], [colonne B])

([colonne A])

()

* Fonction GROUPING([colonne]) : indique la colonne forcée à NULL

0 = fait partie du regroupement

1 = colonne à NULL

* Fonction GROUPING\_ID([colonne A], [colonne B], [colonne C], …) : bits indiquant si une colonne est forcée à NULL.

0 = pas de NULL

2^2 = NULL dans la colonne de droite

2^1

2^0

2^0 + 2^1 + 2^2 = les 3 colonnes sont à NULL

## L2 : Pivoting and Unpivoting Data (p.163)

* **PIVOT**

WITH PivotData AS

(

SELECT

< grouping column >,

< spreading column >,

< aggregation column >

FROM < source table >

)

SELECT < select list >

FROM PivotData

PIVOT( < aggregate function >(< aggregation column >)

FOR < spreading column > IN (< distinct spreading values >) ) AS P;

* **UNPIVOT**

SELECT < column list >, < names column >, < values column >

FROM < source table >

UNPIVOT( < values column > FOR < names column > IN( <source columns> ) ) AS U;

Count(\*) ne fonctionne pas.

1 seule fonction d’aggrégation par pivot.

Les valeurs indiquées dans le « IN » doivent être connues à l’avance.

## L3 : Using Window Functions (p.172)

* Window function s’applique par plage de lignes définies par la clause « OVER »
* Exemple avec la fonction SUM
  + SUM([colonne]) OVER()
  + SUM([colonne]) OVER(PARTITION BY [colonne])
  + SUM([colonne]) OVER(

PARTITION BY [colonne A]

ORDER BY [colonne B], [colonne C]

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW = ROWS UNBOUNDED PRECEDING

(n) ROWS PRECEDING or FOLLOWING

* RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
* **Ranking function**
  + ROW\_NUMBER() = classement au hasard
  + RANK() = 1 2 3 3 5
  + DENSE\_RANK() = 1 2 3 3 4
  + NTILE(100) = nbTotal de lignes / 100
* **Windows offset function**
  + Lag([colonne]) OVER (PARTITION BY … ORDER BY …) = retourne la valeur de la ligne précédente déduite du OVER
  + Lead([colonne]) OVER (PARTITION BY … ORDER BY …) = identique avec la ligne suivante
  + Lag([colonne],[offset]) 🡺 renvoie NULL si hors périmètre
  + First\_Value([colonne]) OVER(PARTITION BY… ORDER BY…ROWS BETWEEN…)
  + Last\_Value([colonne]) OVER(PARTITION BY… ORDER BY…ROWS BETWEEN…)

# Querying and Managing XML Data

## Lesson 1: Returning Results As XML with FOR XML (p.222)

* **Caractères spéciaux :**

& (ampersand) = &amp;

" (quotation mark) = &quot;

< (less than) = &lt;

> (greater than) = &gt;

' (apostrophe) = &apos;  
<![CDATA[...]]>

* **XML prolog (version & encoding) :** <?xml version="1.0" encoding="ISO-8859-15"?>
* **Element centric vs Attribute centric :**

**ATTRIBUTE CENTRIC**

<co:Customer co:custid="1" co:companyname="Customer NRZBB">

<co:Order co:orderid="10692" co:orderdate="2007-10-03T00:00:00" />

<co:Order co:orderid="10702" co:orderdate="2007-10-13T00:00:00" />

<co:Order co:orderid="10952" co:orderdate="2008-03-16T00:00:00" />

</co:Customer>

**ELEMENT CENTRIC**

<co:Customer>

<co:custid>1</co:custid>

<co:companyname>Customer NRZBB</co:companyname>

<co:Order>

<co:orderid>10692</co:orderid>

<co:orderdate>2007-10-03T00:00:00</co:orderdate>

</co:Order>

</co:Customer>

* **Attention à l’ordre des colonnes et du tri pour construire le XML de sortie**
* **FOR XML AUTO**

With XMLNAMESPACES('ns-Order' AS o)

select top 100 [o:OrderHeader].AccountNumber

, [o:OrderHeader].CustomerID

, [o:OrderHeader].DueDate

, [o:OrderDetail].SalesOrderDetailID

, [o:OrderDetail].ProductID

, [o:OrderDetail].LineTotal

from Sales.SalesOrderHeader [o:OrderHeader]

INNER JOIN Sales.SalesOrderDetail [o:OrderDetail] ON [o:OrderDetail].SalesOrderID = [o:OrderHeader].SalesOrderID

Order by [o:OrderHeader].SalesOrderID, [o:OrderDetail].SalesOrderDetailID

For XML AUTO, ELEMENTS, ROOT('Order')

* **FOR XML RAW**

With XMLNAMESPACES('ns-Order' AS o)

select top 100 H.AccountNumber

, H.CustomerID

, H.DueDate

, D.SalesOrderDetailID

, D.ProductID

, D.LineTotal

from Sales.SalesOrderHeader H

INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID

Order by H.SalesOrderID, D.SalesOrderDetailID

For XML RAW

* **FOR XML PATH** = formatage manuel
* **XML Schema Description (XSD)** = describe the metadata of XML documents

select top 100 [OrderHeader].AccountNumber, [OrderHeader].CustomerID

, [OrderDetail].ProductID

, [OrderDetail].LineTotal

from Sales.SalesOrderHeader [OrderHeader]

INNER JOIN Sales.SalesOrderDetail [OrderDetail] ON [OrderDetail].SalesOrderID = [OrderHeader].SalesOrderID

WHERE 1=2

Order by [OrderHeader].SalesOrderID, [OrderDetail].SalesOrderDetailID

For XML AUTO, ELEMENTS, XMLSCHEMA('ns-Order')

* **Mapping fichier XML = table :**Valeurs de OPENXML : 1=attribute centric ; 2=element centric ; 11=mix

DECLARE @DocHandle AS INT;

DECLARE @XmlDocument AS NVARCHAR(1000);

SET @XmlDocument = N'<>....</>'

EXEC sys.sp\_xml\_preparedocument @DocHandle OUTPUT, @XmlDocument

SELECT \*

FROM OPENXML(@DocHandle, '/node1/node2',1)

WITH (attribute1, attribute2)

Exec sys.sp\_xml\_removedocument @DocHandle

Shredding XML to Tables

*Document Object Model* (DOM)

# C08 - Creating Tables and Enforcing Data Integrity

## L1 : Creating and Altering Tables [p.297]

* “Base” table is permanent, table definition and contents remain in the database
* 2 types de création : SELECT INTO ou CREATE TABLE

CREATE TABLE [schema].[name] (

[column] [type] NULL/NOT NULL

)

* Toujours faire référence à la table en utilisant le duo [schema].[table]
* Database schema : conteneur (namespace) utilisé pour regrouper des objets de la base. Il n’y a pas de lien entre les schémas.

CREATE SCHEMA [name] AUTHORIZATION [dbo]

* Pour transférer un objet d’un schéma A vers B :

ALTER SCHEMA [schemaB] TRANSFER [schemaA].[table]

* Limitation à 128 caractères pour le nom des colonnes
* NOT NULL 🡪 ajouter une valeur par défaut

[column] [type] NOT NULL DEFAULT(‘’)

* Identity property : autre moyen de générer un id automatiquement = sequence object

[column] INT IDENTITY(1,1) NOT NULL

Pour forcer l’identité au moment de l’insertion : SET IDENTITY\_INSERT ON

* Computed column : champ calculé à partir d’une expression, si elle est stockée en base, l’expression ne doit pas utiliser de fonction non déterministe (ex. getdate())

[column] AS [expression] [PERSISTED]

* Compression:
  + Row-level compression : a more compact storage format to each row of a table
  + Page-level compression : row-level + algo compression at page-level

CREATE TABLE (…) WITH (DATA\_COMPRESSION = ROW/PAGE);

ALTER TABLE [table] REBUILD WITH(DATA\_COMPRESSION = PAGE)

* + Procédure pour identifier quelle compression utilisé : sp\_estimate\_data\_compression\_savings
* Modification de la table après création : commande ALTER TABLE
* Utiliser le type ROWVERSION au lieu de TIMESTAMP.

## L2 : Enforcing Data Integrity [p.313]

* **Primary Key & Unique**

ALTER TABLE [name] ADD CONSTRAINT [PK\_xxx] PRIMARY KEY([column])

ALTER TABLE [name] ADD CONSTRAINT [UC\_xxx] UNIQUE([column])

Clé primaire : pas de null, pas de doublon dans les données existantes, une seule clé primaire.

Clé unique : null autorisé mais une seule fois

Dans les deux cas, création automatique d’un index. Le nom de la contrainte doit être unique.

* **Foreign key**

ALTER TABLE [name] WITH CHECK

ADD CONSTRAINT [FK\_xxx] FOREIGN KEY([column]) REFERENCES [table] ([column])

Les colonnes liées entre les deux tables doivent avoir le même type de donnée et la même collation.

La colonne référencée doit avoir un index unique (unique, clé primaire ou index).

Il est recommandé de créer un index sur la clé étrangère.

* **Check Constraints**

ALTER TABLE [table] WITH CHECK

ADD CONSTRAINT [CHK\_xxx] CHECK ([column]> =0);

Expression qui doit être validée avant insertion de la donnée, le message d’erreur n’est pas personnalisable.

Attention à la gestion du NULL dans la contrainte.

* **Default Constraints**

CONSTRAINT [DFT\_xxx] [column] DEFAULT(0),

Valeur par défaut au moment d’un insert à la place de NULL.

* **Vues système :** sys.key\_constraints, sys.indexes, sys.foreign\_keys, sys.default\_constraints, sys.check\_constraints

# C09 - Designing and Creating Views, Inline Functions, and Synonyms

## L1 : Designing and Implementing Views and Inline Functions [p.332]

* Objet de base de données constitué d’une commande SELECT sur une ou plusieurs tables ou vues. Requête réutilisable, complexité masquée.

CREATE VIEW [name]

WITH [XXXOPTION]

AS

SELECT …

WITH ENCRYPTION : la requête utilisée dans la vue est cachée aux clients

WITH SCHEMABINDING : la structure des tables utilisées dans la vue ne peuvent pas être modifiée sans suppression de la vue.

WITH VIEW\_METADATA : retourne la structure correspondante au résultat de la vue

WITH CHECK OPTION : contrôle que la mise à jour de la table vérifie bien la clause WHERE de la vue.

* Commande de création/mise à jour : DROP VIEW/ALTER VIEW
* Catalog view : sys.views
* Partitioned view = assemblage de données stockées sur plusieurs tables
  + Local partitioned view : union sur des tables stockée dans la même base
  + Distributed partitioned view : union sur des tables réparties sur plusieurs instances de BDD
* **Inline function** = simule une vue paramétrée

CREATE FUNCTION [name] ([@parameter] [type])

RETURNS TABLE

AS

RETURN

(

SELECT…FROM..

WHERE … = [@parameter]

)

## L2 : Using Synonyms [p.347]

* Noms stockés en base et utilisés pour se substituer au nom des objets dans les requêtes.

CREATE SYNONYM [name] FOR [origin\_path]

* L’existence de l’objet “bindé” est testée au moment du requêtage.
* Utilisable sur les objets suivants : tables (+temporaire), vues, proc, fonctions.
* Pas d’ALTER mais DROP uniquement
* 1 synonyme ne peut être substitué à 1 autre synonyme.
* Pas de donnée stockée, ni de code associé au synonyme.

# C10 - Inserting, Updating, and Deleting Data

## L1 : Inserting Data [p.363]

* Commandes INSERT :

INSERT INTO [table] (col1,col2,col3) VALUES (val1,val2,val3), (val1’,val2’,val3’), (val1”,val2”,val3”)

INSERT INTO [table] SELECT …

INSERT INTO [table] EXEC…

* Désactivation de la contrainte d’identité :

SET IDENTITY\_INSERT [table] ON/OFF

* Mot clé “DEFAULT” à mettre dans les valeurs à insérer
* SELECT [x colonnes] INTO [tabledest] FROM [tablesource]
  + attention les contraintes, les index, les triggers ne sont pas recopiés
  + minimal logging
  + lock data + metadata

## L2 : Updating Data [p.373]

* Commande UPDATE :

UPDATE [table] SET [@var] = [colonne] = [value]

* Possibilité d’utiliser des jointures et des CTE (attention aux requêtes non déterministes) : 1 seule table updatée

## L3 : Deleting Data [p.388]

* Commande DELETE :

DELETE FROM [table] WHERE [predicat]

Si gros volume de données à supprimer, il est conseillé de les supprimer par paquet pour éviter les locks

DELETE TOP(x) FROM [table]

* DELETE vs TRUNCATE
  + + de log, enregistre l’élément supprimé, + lent || Enregistre la page, - log, + rapide
  + Pas de raz de l’identity || raz identity
  + Autorisé sur une vue indexée || non autorisé
  + Droits de DELETE || Droits d’ALTER
  + Si clé étrangère sans lien : autorisé || Non autorisé
* Table Expression :
  + Utiliser une CTE pour supprimer des données dans un ordre précis par exemple :

WITH CTE AS (SELECT TOP(10) \* FROM [table]) DELETE FROM CTE

# C11 - Other Data Modification Aspects

## L1 - Using the Sequence Object and IDENTITY Column Property [p.402]

* Identity property : applicable 1 fois par table sur une colonne numérique

[column] INT NOT NULL IDENTITY(seed,increment) CONSTRAINT [PK\_name] PRIMARY KEY;

* Forçage de l’id à l’insertion :

SET IDENTITY\_INSERT [table] ON

* Fonctions :

SCOPE\_IDENTITY() 🡪 identité créée dans la session

@@IDENTITY 🡪 dernière identité créée dans la session sur les scopes

IDENT\_CURRENT(‘[table]’) 🡪 dernière identité pour une table donnée (sans restriction)

* Commande pour remettre à niveau le « seed » après un truncate par ex :

DBCC CHECKIDENT(‘[table]’,RESEED,4)

* Désavantages de l’IDENTITY :

Attention, unicité non garantie (ex : saisie de la valeur avec IDENTITY\_INSERT)

Suite numérique non garantie (ex : erreur lors d’un insert, le compteur est quand même incrémenté)

Pas de boucle si la valeur max est atteinte

* L’objet Séquence n’est pas lié à une propriété d’une colonne

CREATE SEQUENCE [schema].[objet] AS [type]

INCREMENT BY [x]

MINVALUE [x]

MAXVALUE [x]

CYCLE/NO CYCLE

START WITH (par défaut =MINVALUE)

* Commande pour obtenir une nouvelle valeur d’une séquence :

SELECT NEXT VALUE FOR [sequence]

* Commande pour redémarrer une séquence :

ALTER SEQUENCE [sequence] RESTART WITH [value]

* Mettre par défaut la prochaine valeur d’une séquence dans une colonne (possibilité de supprimer cette contrainte par la suite) :

ADD CONSTRAINT [DF\_name] DEFAULT(NEXT VALUE FOR [sequence]) FOR [colonne]

* Générer l’id dans le SELECT d’insertion :

SELECT NEXT VALUE FOR [sequence] OVER( ORDER BY [column]) FROM [table]

* Option CACHE [value] :

Mémorise les prochaines valeurs de la séquence, plus performant mais risque de gap en cas de plantage.

## L2 - Merging Data [p.415]

* Commande Merging :

MERGE INTO [target] WITH (SERIALIZABLE|HOLDLOCK) AS [TGT]

USING [source] AS [SRC]

ON [predicate]

WHEN MATCHED {and predicate}

THEN UPDATE SET/DELETE…

WHEN NOT MATCHED {BY TARGET} {and predicate}

THEN INSERT VALUES (…)

WHEN NOT MATCHED BY SOURCE {and predicate}

THEN UPDATE/DELETE…

* Pour éviter les conflits d’insertion/mise à jour entre deux sessions :

WITH (SERIALIZABLE | HOLDLOCK)

* Attention la clause ON ne filtre par les données !

## L3 - Using the OUTPUT Option [p.393]

Retourne les informations sur les lignes qui ont été modifiées.

Possibilité d’utiliser deux instruction OUTPUT pour une même requête : l’une avec INTO et l’autre sans.

* INSERT

INSERT INTO [table] ([column1], [column2], …)

OUTPUT

inserted.[column1], inserted.[column2], …

{INTO [table]}

SELECT …

* DELETE

DELETE FROM [table]

OUTPUT deleted.[column1], …

{INTO [table]}

WHERE …

* UPDATE

UPDATE [table]

SET [column1] = [value1]

OUTPUT

inserted.column1 AS old

, deleted.column1 AS new

{INTO [table]}

* MERGE

MERGE …

OUTPUT

$action

, COALESCE(inserted.[column1], deleted.[column1]) as [column1]

Fonction $action pour determiner quelle action a été réalisée par le « merge ».

Dans le cas de l’utilisation du composable DML (INSERT INTO… (MERGE .. OUTPUT …)), la table de destination ne doit pas avoir de clé étrangère et trigger.

# C12 - Implementing Transactions, Error Handling, and Dynamic SQL [p.443]

## L1 : Managing Transactions and Concurrency [p.444]

* ACID = Atomicity Consistency Isolation Durability
* COMMIT TRAN/WORK - ROLLBACK TRAN/WORK
* @@TRANCOUNT:
  + >0 : transaction active
  + N’indique pas les transactions non commitables
* XACT\_STATE():
  + 0 : no active transaction
  + 1 : uncommitted transaction
  + -1 : fatal error, uncommitted transaction
  + N’indique pas les transactions imbriquées
* Transaction mode :
  + Autocommit mode : mode par défaut, commit à chaque DML ou DDL
  + Implicit transaction : le « begin tran » est automatique mais penser à faire le « commit » ou « rollback »
  + Explicit transaction : begin tran + commit tran/rollback tran
  + Transactions imbriquées:

1 rollback invalide toutes les transactions. Le commit intermédiaire sert juste à diminuer le @@TRANCOUNT

* Marquer une transaction :

BEGIN TRAN [transaction\_name] WITH MARK ;

Place le nom de la transaction dans le journal des transactions, utilisé dans le cadre d’une restauration de base de données.

RESTORE DATABASE [database] FROM DISK = ‘[path]’

WITH STOPATMARK|STOPBEFOREMARK = ‘[transaction\_name]’

* Point de sauvegarde à l’intérieur d’une transaction

If @startingTrancount > 0

SAVE TRANSACTION [transaction\_name]

ELSE

BEGIN TRANSACTION

[…]

ROLLBACK TRAN [transaction\_name]

* Transaction sur plusieurs bases de données : en cas de failover sur une des bases, la transaction est corrompue.
* Transaction distribuée : requêtes sur plusieurs serveurs
* Basic Locking
  + **Shared locks** = readers 🡪 lecture partagée, écriture interdite
  + **Exclusive locks** = writers 🡪lecture et écriture interdites
  + 3 niveaux de locks : pages, tables ou lignes
  + DeadLock : deux transactions travaillent sur la même ressource.
* Transaction Isolation levels : (à forcer dans chaque session)
  + **Read committed** (par défaut) 🡪 attente de la libération de la ressource avant lecture
  + **Read uncommitted** 🡪 le shared lock est supprimé, lecture de données non commitées
  + **Read committed snapshot** (= RCSI) (par défaut sur windows Azure)

Défini au niveau de la BDD et non sur la session

Enregistre dans tempdb les valeurs originales avant transaction

* Repeatable read 🡪 par session
* Snapshot
* Serializable

Utilisation des paramètres « WITH NOLOCK » ou « WITH (READUNCOMMITTED)” pour forcer la lecture d’une table lorsqu’on est en mode « READ COMMITTED » 🡪 attention à la qualité de la donnée

## L2 : Implementing Error Handling [p.467]

* **Error number :**

1 à 49 999 : sql error

50 000 : custom error sans numéro

> 50 000 : custom error

* **Severity level :** 0 à 25

>16 : enregistré dans le log SQL server et Windows

19 à 25 : spécifié uniquement par sysadmin

20 à 25 : fatal error = connection closed + transaction rollbackée

0 à 10 : information

* **State :**

maximum 127 – Information interne à SQL server

* **Error message :** listé dans sys.messages + custom message (sp\_addmessage)
* **RAISERROR( [message|message\_id|@msg\_variable], severity, state)**

Affichage d’info comme un print = severity level entre 0 et 9

Génération d’une erreur puis fin de connexion : severity > 20 + WITH LOG + sysadmin role

WITH OPTION NO WAIT : envoi immédiat du message au client

* **THROW error\_number, message, state ;**

Pas de NOWAIT, ni de WITH LOG

L’erreur\_number n’est pas obligatoirement défini la table sys.messages.

Peut-être utilisé sans paramètre dans un bloc CATCH pour transmettre l’erreur sans modification au client.

Sévérité toujours paramétré à 16

Termine le batch sauf en cas de TRY/CATCH

La commande précédent le THROW doit se terminer par « ; »

* **FORMATMESSAGE**(‘[message]%[message],’[string\_to\_insert]’)
* **TRY\_CONVERT** (type, data) – TRY\_PARSE(‘string’ AS Type)

Renvoient NULL si la conversion est impossible

* @@ERROR = error number
* **XACT\_ABORT :** si la sévérité de l’erreur est > à 10, la transaction est rollbackée.
* **TRY/CATCH**

1 RAISERROR de sévérité de 11 à 19 n’envoie pas de message au client mais renvoie au bloc CATCH

Les erreurs avec sévérité > 19 sont envoyées au CATCH si pas de déconnexion

Ne gère pas les erreurs fatales

* Obtenir plus d’informations : ERROR\_NUMBER(), ERROR\_MESSAGE(), ERROR\_LINE(), ERROR\_SEVERITY(), ERROR\_STATE()

## L3: Using Dynamic SQL [p.482]

Impossible de modifier directement : noms de colonne, table, database avec le Use, les listes IN et Pivot 🡺 utilisation du code dynamique.

Technique qui consiste à utiliser du code SQL pour générer et executer un autre code SQL.

Exemple d’utilisation : une procédure qui comporte un SELECT avec plusieurs filtres paramétrables, la commande PIVOT

* SET QUOTE\_IDENTIFIER ON : défini les ‘ pour les strings et les ‘’ pour les commandes SQL
* Fonction pour doubler les quotes : QUOTENAME(N’[query]’,’’’’)
* EXEC(‘[single batch]’)
* Injonction de dépendance : insertion d’une quote entre le quote final et le quote de fermeture d’une string.
* EXEC sp\_executesql @statement=’’,@params=’’,@param1=’’
  + solution pour éviter l’injonction de dépendance
  + possibilité d’avoir des paramètres de sortie
  + chaîne unicode en entrée

# C13 - Designing and Implementing T-SQL Routines

## L1 : Designing and Implementing Stored Procedures [p.502]

* Quelques avantages: optimisation (plans d’exécution enregistrés), encapsulation du SQL, sécurité

CREATE PROC [name]

@var1 AS [type]

, @var2 AS [type] = 0 OUTPUT

AS

BEGIN

…

SET @var2 = [data];

RETURN;

END

DECLARE @varOutput AS type

EXEC [name]

@var1 = [data]

, @var2 = @varOutput OUTPUT;

* Test d’existence : IF OBJECT\_ID(‘[procedure\_name]’,’P’) IS NOT NULL
* SET NOCOUNT ON 🡪 désactive le message indiquant le nombre de lignes affectées par une requête (moins de trafic client-server)
* IF/ELSE
* WHILE – BREAK (sortir de la boucle) – CONTINUE (revenir au début de la boucle)
* WAITFOR DELAY ‘xx:xx:xx’ : pause pendant x seconds
* WAITFOR TIME ‘xx:xx:xx’ : démarrage à une heure précise
* GOTO : jump jusqu’à un label
* Résultats d’une procédure : variable OUTPUT, le RETURN et les SELECT

## L2 : Implementing Triggers [p.522]

* Sorte de procédure stockée associée à un évènement DML (data manipulation langage) sur une table ou une vue.
* **AFTER Triggers :** défini uniquement pour les tables, est exécuté à la suite d’une commande de type DML et après toutes les contraintes.

Après une commande DELETE ou UPDATE, commencer le trigger par un test sur le @@ROWCOUNT pour éviter d’exécuter des traitements pour rien.

Utiliser les tables « deleted » et « inserted », pour tester les valeurs qui ont été modifiées.

CREATE TRIGGER TriggerName ON [dbo].[TableName]

[FOR|AFTER] DELETE, INSERT, UPDATE

AS

BEGIN

IF @@ROWCOUNT = 0 RETURN;

SET NOCOUNT ON

END

Mauvaise pratique : retourner un ensemble de résultats par le trigger.

Nombre maximum de trigger imbriqués ou exécutés en « boucle » : 32

Désactivation : exec sp\_configure ‘nested trigger’, 0 + reconfigure

* **INSTEAD OF :** peut être exécuté sur une table ou une vue en remplacement d’une action DML.
* **DML trigger function :**

UPDATE([colonne\_name]) booléen pour savoir si une colonne est concernée par une maj.

COLUMNS\_UPDATED() : retourne un modèle binaire de type varbinary qui indique les colonnes d'une table ou d'une vue qui ont été insérées ou mises à jour.

## Lesson 3: Implementing User-Defined Functions

C14 - Using Tools to Analyze Query Performance

L1 – Getting Started With Query Optimization

* TSQL 🡪 Parsing (check syntax => parse tree) 🡪 Binding (object resolution => algebrized tree) 🡪 Optimization (generate candidate plan and select one => execution plan, single tree with physical operators) 🡪 Execution (query exec + plan caching)
* Relational engine = work on a logical level
* Storage engine = execution part, carry out the physical operations
* Query optimizer = transformation from logical to physical operators
* High cost > more complex plan > slower query
* Query optimizer cannot guarantee that the best possible plan is selected (hundred combination to test !)
* Cost : number of rows that have to be processed (cardinality estimation)
* Caching : next execution will be faster but warning obsolete plan
* Wrong scenarios : trop de possibilités dans le périmètre de plan d’exécution à définir, statistics non présents ou trop vieux, parameter sniffing (mauvaise prévision => coût mal calculé)

## XML

XML adapté pour l’envoi de messages entre applications, auto-descriptif, représentation des données sous forme d’arborescence.

Colonne de type XML : stockage du document XML dans un format binaire (blob – max 2Go – max 128 niveaux – UTF-16).

XML typé : association d’une collection de schémas à la colonne XML (contrainte d’intégrité forte)

XML non typé = document XML 1.0 bien formé ou fragment XML

* Définition d’un schéma

CREATE XML SCHEMA COLLECTION nomDuSchema

AS ‘DéfinitionDuSchema’

SELECT xml\_schema\_namespace(N'TableName',N'SchemaCollectionName')

* Définition d’une colonne s’appuyant sur le schéma

CREATE TABLE tblName (

…

colExemple xml (nomDuSchema)

) ;

* Balise **any** dans le schéma : champ optionnel non défini à l’avance

<xs:complexType name=”client”>

<xs:sequence>

<xs:element name=”nom”/>

<xs:element name=”prenom/>

<xs :any namespace=”##other” processContents=”skip” minOccurs=”0” maxOccurs=”unbounded”/>

</xs:sequence>

</xs:complexType>

* Exemple de schéma avec liste de valeur et union

<xs :simpletType name= « Unite »>

<xs :union>

<xs:simpleType>

<xs:list>

<xs:simpleType>

<xs:restriction base=”xs:string”>

<xs:enumeration value=”cm”/>

<xs:enumeration value=”m”/>

<xs:enumeration value=”km”/>

</xs:restriction >

</xs:simpleType>

</xs:list>

</xs:simpleType>

<xs:simpleType>

<xs:list>

<xs:simpleType>

<xs:restriction base=”xs:string”>

<xs:enumeration value=”foot”/>

<xs:enumeration value=”m”/>

</xs:restriction >

</xs:simpleType>

</xs:list>

</xs:simpleType>

</xs :union>

</xs :simpletType>

* Ajout d’une ligne dans une colonne XML non typée :

INSERT INTO tableName (col1,colXml)

VALUES(1,’<page>….</page>’)

* Méthode SQL spécifiques
  + **query()** : retourne un élément XML non typé

SELECT colXML.query(‘noeud1/noeud2/noeud3’) FROM tableName

* + **value() :** retourne une valeur scalaire
  + **exist() :** vérifie l’existence d’un nœud

SELECT colXML.value(‘(noeud1/noeud2/noeud3)[1]’,’typeDonnee’)

FROM tableName

WHERE colXML.exist(‘(noeud1/noeud2/noeud3)[1]’)=1

* + **node()**: à partir d’une requête XQuery passée en paramètre, renvoie autant de lignes que de nœuds définis à ce niveau

SELECT resultats.x.query(‘.’) from @xmlFile.nodes(‘/page/article’) as resultat(x);

* + **modify()** : permet de modifier une partie des données stockées dans un document XML

Declare @nouvelArticle as XML ;

Set @nouvelArticle = ‘<article>i10</article>’

Set @xmlDoc.modify(‘insert sql :variable(“@nouvelArticle”) as last into (/page)[1]’);

* **Indexer une colonne de type XML**
  + Index principal (index classique sous forme d’arbre)

CREATE PRIMARY XML INDEX nomIndex ON nomTable(xmlColonne);

* + Index secondaire
    - Path : requête portant sur le chemin d’accès
    - Property : requête portant sur les propriétés
    - Value : requête portant sur des valeurs

CREATE XML INDEX nomIndex ON nomTable(colonneXML)

USING XML INDEX nomIndexPrincipal FOR {Path|Property|Value}

* **XQuery** et **XPath**
  + XQuery = langage pour interroger des données au format XML
  + XPath 2.0 = langage de navigation intégré dans XQuery, requêtes XQuery définis uniquement avec un chemin

SELECT @xmlDoc.query(‘racine/noeud1’);

SELECT @xmlDoc.query(‘racine/noeud1/\*’);

SELECT @xmlDoc.query(‘racine/noeud1[1]/\*’);

SELECT @xmlDoc.query(‘racine/noeud1[1]/@champ1’,’typeValeur’); 🡪renvoie la valeur contenue dans le champ interrogé

SELECT @xmlDoc.query(‘racine/noeud1[@champ1=’’value’’]/\*’); 🡪 sélectionne un nœud en fonction de value

SELECT @xmlDoc.query(‘

<root>{

for $element in /rootDoc/noeud1

let $nombre:=count($element/noeud2)

return

<toto>

{$element/@champ1}

<nbElements>{$nombre}</nbElements>

</toto>

}</root>’);

* **FOR XML**

Conversion au format XML d’un résultat d’une requête SELECT

SELECT …

FROM …

FOR XML AUTO[, TYPE] ou FOR XML PATH(‘root’),TYPE

* **OpenXML**

Méthode qui permet de traiter un document XML sous la forme d’un jeu de résultat (tend à être remplacer par le type XML)

* + sp\_xml\_preparedocument : prépare un document au format texte vers le format XML. Renvoie un identifiant utilisé par OpenXML pour localiser les données.
  + sp\_xml\_removedocument : libère l’espace mémoire
  + OPENXML(idoc int, requeteXPath nvarchar, WITH (structureDonnées))

Exec sp\_xml\_preparedocument @identifiantDoc out, @strDoc

Select \* from openxml(@identifiantDoc,’root/\*’) with(champ1 type1, champ2 type2);

Exec sp\_xml\_removedocument @identifiantDoc

* **OpenRowset**

Méthode qui permet d’accéder facilement à des ressources externes au serveur pour travailler avec ces données distantes comme si elles étaient présentes sous forme de table (fichier, OLDEDB) – peut remplacer BCP.

SELECT \* FROM OPENROWSET(BULK ‘c:\file.xml’, SINGLE\_BLOB) AS information

http://sqlwithmanoj.wordpress.com/xml-sql/