Contents

[Filtering and Sorting Data 3](#_Toc349858332)

[Lesson 1: Filtering Data with Predicates (p.62) 3](#_Toc349858333)

[Lesson 2: Sorting Data 3](#_Toc349858334)

[Lesson 3: Filtering Data with TOP and OFFSET-FETCH 3](#_Toc349858335)

[Combining Sets 3](#_Toc349858336)

[Lesson 1: Using Joins (p.102) 3](#_Toc349858337)

[Lesson 2: Using Subqueries, Table Expressions, and the APPLY Operator (p.117) 3](#_Toc349858338)

[Lesson 3: Using Set Operators (p.136) 4](#_Toc349858339)

[Grouping and Windowing 4](#_Toc349858340)

[Lesson 1: Writing Grouped Queries (p.150) 4](#_Toc349858341)

[Lesson 3: Using Window Functions (p.172) 6](#_Toc349858342)

[Creating Tables and Enforcing Data Integrity 7](#_Toc349858343)

[Lesson 1: Creating and Altering Tables 7](#_Toc349858344)

[Lesson 2: Enforcing Data Integrity (p.281) 7](#_Toc349858345)

[Designing and Creating Views, Inline Functions, and Synonyms 8](#_Toc349858346)

[Lesson 1: Designing and Implementing Views and Inline Functions (p.300) 8](#_Toc349858347)

[Lesson 2: Using Synonyms (p.135) 8](#_Toc349858348)

[Inserting, Updating, and Deleting Data 9](#_Toc349858349)

[Lesson 1: Inserting Data (p.330) 9](#_Toc349858350)

[Lesson 2: Updating Data (p.341) 9](#_Toc349858351)

[Lesson 3: Deleting Data (p.356) 9](#_Toc349858352)

[Other Data Modification Aspects 9](#_Toc349858353)

[Lesson 1: Sequence Object and IDENTITY Column Property (p.370) 9](#_Toc349858354)

[Lesson 2: Merging Data (p.382) 10](#_Toc349858355)

[Lesson 3: Using the OUTPUT Option (p.393) 10](#_Toc349858356)

[Implementing Transactions, Error Handling, and Dynamic SQL 10](#_Toc349858357)

[Lesson 1: Managing Transactions and Concurrency (p.412) 10](#_Toc349858358)

[Lesson 2: Implementing Error Handling (p.435) 11](#_Toc349858359)

[Lesson 3: Using Dynamic SQL (p.450) 12](#_Toc349858360)

[Designing and Implementing T-SQL Routines 13](#_Toc349858361)

[Lesson 1: Designing and Implementing Stored Procedures (p.470) 13](#_Toc349858362)

[Lesson 2: Implementing Triggers (p.490) 13](#_Toc349858363)

[Lesson 3: Implementing User-Defined Functions 13](#_Toc349858364)

# Filtering and Sorting Data

## Lesson 1: Filtering Data with Predicates (p.62)

* 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

## Lesson 2: Sorting Data (p.74)

* 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.

## Lesson 3: Filtering Data with TOP and OFFSET-FETCH (p.84)

* 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

# Combining Sets

## Lesson 1: Using Joins (p.102)

* 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.

## Lesson 2: Using Subqueries, Table Expressions, and the APPLY Operator (p.117)

* 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.

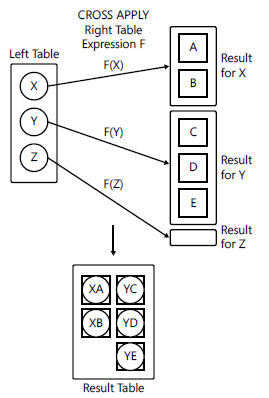
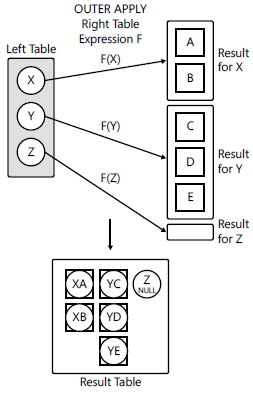
CREATE FUNCTION <function> (<@var> AS INT) RETURNS TABLE

AS

RETURN

SELECT … FROM <table>;

* CROSS APPLY – OUTER APPLY

## Lesson 3: Using Set Operators (p.136)

* 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

# Grouping and Windowing

## Lesson 1: Writing Grouped Queries (p.150)

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

SELECT <col>

FROM <table>

GROUP BY GROUPING SETS

(

(<colA>,<colB>),

(<colB>)

)

* GROUP BY CUBE(<colA>,<colB>) équivalent à :

(<col1>,<col2>)

(<col1>)

(<col2>)

()

* GROUP BY ROLLUP(<col1>,<col2>,<col3>) équivalent à :

(<col1>,<col2>,<col3>)

(<col1>,<col2>)

(<col1>)

()

* Fonction GROUPING(<colonne>) : indique la colonne forcée à NULL

0 = fait partie du regroupement

1 = colonne à NULL

* Fonction GROUPING\_ID(<colonne1>,<colonne2>,<colonne3>,…) : 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

**Lesson 2: 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.

## Lesson 3: Using Window Functions (p.172)

* Window function s’applique par plage de lignes définis par la clause « OVER »
* Exemple avec la fonction SUM
  + SUM(<col>) OVER()
  + SUM(<col>) OVER(PARTITION BY <col>)
  + SUM(<col>) OVER(

PARTITION BY <colA>

ORDER BY <colB>, <colC>

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(<col>) OVER (PARTITION BY … ORDER BY …) = retourne la valeur de la ligne précédente déduite du OVER
  + Lead(<col>) OVER (PARTITION BY … ORDER BY …) = identique avec la ligne suivante
  + Lag(<col>,<offset>) 🡺 renvoie NULL si hors périmètre
  + First\_Value(<col>) OVER(PARTITION BY… ORDER BY…ROWS BETWEEN…)
  + Last\_Value(<col>) OVER(PARTITION BY… ORDER BY…ROWS BETWEEN…)

# Querying and Managing XML Data

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

& (ampersand) &amp;

" (quotation mark) &quot;

< (less than) &lt;

> (greater than) &gt;

' (apostrophe) &apos;

<![CDATA[...]]>

<?xml version="1.0" encoding="ISO-8859-15"?>.

* Element centric VS 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>

<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>

WITH XMLNAMESPACES('TK461-CustomersOrders' AS co)

SELECT [co:Customer].custid AS [co:custid],

[co:Customer].companyname AS [co:companyname],

[co:Order].orderid AS [co:orderid],

[co:Order].orderdate AS [co:orderdate]

FROM Sales.Customers AS [co:Customer]

INNER JOIN Sales.Orders AS [co:Order]

ON [co:Customer].custid = [co:Order].custid

WHERE [co:Customer].custid <= 2

AND [co:Order].orderid %2 = 0

ORDER BY [co:Customer].custid, [co:Order].orderid

FOR XML AUTO, ELEMENTS, ROOT('CustomersOrders');

XML Schema Description (XSD) = describe the metadata of XML documents

FOR XML RAW : creating XML from a query result is the RAW option

FOR XML RAW, ELEMENTS

FOR XML AUTO

FOR XML AUTO, ELEMENTS, ROOT('CustomersOrders');

FOR XML AUTO, ELEMENTS, XMLSCHEMA('TK461-CustomersOrders'); : afficher le schéma XSD

Shredding XML to Tables

*Document Object Model* (DOM)

# Creating Tables and Enforcing Data Integrity

## Lesson 1: Creating and Altering Tables

* CREATE TABLE <name> (

<colonne> <type> NULL/NOT NULL

)

* Database schema : conteneur (namespace) utilisé pour regrouper des objets de la base

Pas de relation entre schéma

* CREATE SCHEMA <name> AUTHORIZATION <dbo>
* ALTER SCHEMA <name> TRANSFER <table>
* NOT NULL 🡪 ajouter une valeur par défaut

<colonne> <type> NOT NULL DEFAULT(‘’)

* Identity :

<colonne> INT IDENTITY(1,1) NOT NULL

* Colonne calculée :

<colonne> AS ColA \* ColB [PERSISTED]

* Compression:

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

ALTER TABLE <name> REBUILD WITH( DATA\_COMPRESSION = PAGE)

sp\_estimate\_data\_compression\_savings : aide à déterminer quelle compression est à utiliser.

## Lesson 2: Enforcing Data Integrity (p.281)

* ALTER TABLE <name> ADD CONSTRAINT PK\_... PRIMARY KEY(<col>)
* ALTER TABLE <name> ADD CONSTRAINT UC\_... UNIQUE(<col>)

1 NULL est accepté avec UNIQUE.

* **Foreign key**

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)

ALTER TABLE <name> WITH CHECK

ADD CONSTRAINT FK\_... FOREIGN KEY(<col>) REFERENCES <table> (<col>)

* **Check Constraints :** expression qui doit être validée avant insertion de la donnée, le message d’erreur ne peut être personnalisé.

ALTER TABLE <table> WITH CHECK

ADD CONSTRAINT CHK\_... CHECK (<col> =0);

* **Default Constraints :** valeur par défaut au moment d’un insert à la place de NULL

CONSTRAINT DFT\_Products\_unitprice DEFAULT(0),

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

# Designing and Creating Views, Inline Functions, and Synonyms

## Lesson 1: Designing and Implementing Views and Inline Functions [p.300]

* 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 …

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 : option utilisée dans le cas d’une mise à jour de table à partir d’une vue.

* DROP VIEW/ALTER VIEW
* sys.views : catalog view
* Partitioned view = requête de type UNION interrogeant des tables stockées sur différents serveurs.
* Inline function = simule une vue paramétrée

CREATE FUNCTION <name> (<@paramètre> <type>)

RETURNS TABLE

AS

RETURN

(

SELECT…FROM..

WHERE … = <@paramètre>

)

## Lesson 2: Using Synonyms (p.135)

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

CREATE SYNONYM <syn> FOR <origin>

Pas d’ALTER mais DROP uniquement

1 synonyme ne peut être substitué à 1 autre synonyme.

# Inserting, Updating, and Deleting Data

## Lesson 1: Inserting Data (p.330)

* INSERT INTO <tbl> VALUES…
* INSERT INTO <tbl> SELECT…
* INSERT INTO <tbl> EXEC…
* SET IDENTITY\_INSERT <tbl> ON/OFF
* Keyword “DEFAULT” à mettre dans les valeurs à insérer
* SELECT <col> INTO <tbl> FROM <tbl> 🡪attention les contraintes ne sont pas recopiées.

## Lesson 2: Updating Data (p.341)

* UPDATE <tbl> SET <@var> = <col> = <value>
* Possibilité d’utiliser des jointures et des CTE

## Lesson 3: Deleting Data (p.356)

* DELETE FROM <tbl> WHERE…
* DELETE vs TRUNCATE
  + + de log, enregistre l’élément supprimé / Enregistre la page
  + 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

# Other Data Modification Aspects

## Lesson 1: Sequence Object and IDENTITY Column Property (p.370)

* <col> INT NOT NULL IDENTITY(1,1)

CONSTRAINT <PK\_> PRIMARY KEY,

* SET IDENTITY\_INSERT <tbl> ON/OFF
* SCOPE\_IDENTITY() = identité créée dans la session
* @@IDENTITY
* IDENT\_CURRENT(‘<tbl>’)
* DBCC CHECKIDENT(‘<tbl>’,RESEED,4)
* Désavantages de l’IDENTITY :

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

Suite numérique non garantie (ex : plantage à l’insert)

Pas de boucle

* CREATE SEQUENCE <schema>.<objet> AS <Type>
  + INCREMENT BY <x>
  + MINVALUE <x>
  + MAXVALUE <x>
  + CYCLE/NO CYCLE
  + START WITH (par défaut =MINVALUE)
* Nouvelle valeur : SELECT NEXT VALUE FOR <seq>
* Redémarrer une séquence : ALTER SEQUENCE <seq> RESTART WITH <x>
* Mettre par défaut la prochaine valeur d’une séquence dans une colonne :

DEFAULT(NEXT VALUE FOR <seq>) FOR <col>

SELECT NEXT VALUE FOR <seq> OVER( ORDER BY <col>) FROM <tbl>

## Lesson 2: Merging Data (p.382)

MERGE INTO <tbl> WITH (HOLDLOCK) AS [TGT]

USING <tbl> AS [SRC]

ON <predicate>

WHEN MATCHED [and predicate]

THEN UPDATE SET …

WHEN NOT MATCHED BY TARGET [and predicate]

THEN INSERT VALUES (…)

WHEN NOT MATCHED BY SOURCE [and predicate]

THEN <action>

WITH (HOLDLOCK) permet d’éviter les conflits d’insertion/mise à jour entre deux sessions.

## Lesson 3: Using the OUTPUT Option (p.393)

* INSERT

INSERT INTO <tbl> ()

OUTPUT

Inserted.value1, inserted.value2, …

INTO <tbl>

SELECT …

* DELETE

DELETE FROM <tbl>

OUTPUT deleted.value1

WHERE …

* UPDATE

UPDATE <tbl>

SET …

OUTPUT

Inserted.value1 AS old

, deleted.value1 AS new

* MERGE

MERGE …

OUTPUT $action

, COALESCE(inserted.value1, deleted.value1) as value1

# Implementing Transactions, Error Handling, and Dynamic SQL

## Lesson 1: Managing Transactions and Concurrency (p.412)

* ACID = Atomicity Consistency Isolation Durability
* COMMIT TRAN/WORK - ROLLBACK TRAN/WORK
* @@TRANCOUNT:
  + >0 : transaction active
* XACT\_STATE():
  + 0 : no active transaction
  + 1 : uncommitted transaction
  + -1 : fatal error, uncommitted transaction
* Transaction mode :
  + Autocommit mode
  + Implicit transaction : indiquer seulement 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 <nom> WITH MARK ;

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

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

If @startingTrancount > 0

SAVE TRANSACTION <name>

ELSE

BEGIN TRANSACTION

[…]

ROLLBACK TRAN <name>

* Transaction distribuée : requêtes sur plusieurs serveurs
* Basic Locking
  + Shared locks = readers 🡪 lecture partagée, écriture interdite
  + Exclusive locks = writers 🡪lecture et écriture interdites
* Transaction Isolation levels
  + Read committed (par défaut)
  + Read uncommited
  + 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
* Snapshot
* Serializable

## Lesson 2: Implementing Error Handling (p.435)

* 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

20 à 25 : fatal error

0 à 10 : information

* State :

maximum 127 – Information interne à SQL server

* Error message (sys.messages)
* RAISERROR(‘msg’, severity, state)

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

* FORMATMESSAGE(‘msg%msg’,’msg’)
* Option NO WAIT pour envoyer immédiatement les messages au client
* THROW error\_number, mesage, state ;

Termine le batch

Pas de NOWAIT, ni de WITH LOG

* TRY\_CONVERT (type, data) – TRY\_PARSE(‘string’ AS Type)

Renvoient NULL si impossible de convertir

* @@ERROR = error number
* XACT\_ABORT : en cas d’erreur, la transaction est stoppée et annulée automatiquement – dans un TRY/CATCH, renvoie au bloc CATCH
* TRY/CATCH

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

Ne gère pas les erreurs fatales

Utiliser la commande THROW sans paramètre dans le bloc CATCH pour obtenir le message d’erreur d’origine.

## Lesson 3: Using Dynamic SQL (p.450)

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
* EXEC(‘msg’) : attention au SQL injection
* EXEC sp\_executesql @statement=’’,@params=’’,@param1=’’

# Designing and Implementing T-SQL Routines

## Lesson 1: Designing and Implementing Stored Procedures (p.470)

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;

* IF/ELSE
* WHILE – BREAK – CONTINUE
* WAITFOR DELAY ‘xx:xx:xx’ : pause pendant x seconds
* WAITFOR TIME ‘xx:xx:xx’ : démarrage à une heure précise
* GOTO
* Résultats d’une procédure : variable OUTPUT, le RETURN et les SELECT

## Lesson 2: Implementing Triggers (p.490)

* **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 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

* **INSTEAD OF :** peut être exécuté sur une table ou une vue
* **DML trigger function :**

UPDATE(<col>) booléen pour savoir si une colonne était concernée par la maj.

COLUMNS\_UPDATED()

## L3 : Implementing User-Defined Functions [p.533]

* Encapsuler du code TSQL réutilisable et retourner une valeur scalaire ou une table
* Accepte les paramètres, les UDFs sont exécutées dans une commande TSQL
* FN = SQL scalar function = retourne une seule valeur

IF OBJECT\_ID(‘*fn\_function*’,’FN’) IS NOT NULL

DROP FUNCTION *fn\_function*

CREATE FUNCTION *functionName*

(

*@param1 int,*

*@param2 int*

)

RETURNS INT

AS

BEGIN

RETURN *@param1 + @param2*

END

* IF = SQL inline table-valued function

CREATE FUNCTION *dbo.FunctionName*

(

*@param1 int,*

*@param2 char(5)*

)

RETURNS TABLE AS RETURN

(

*SELECT @param1 AS c1,*

*@param2 AS c2*

)

Bloc BEGIN/END non obligatoire.

* TF = SQL table-valued-function

CREATE FUNCTION *dbo.FunctionName*

(

*@param1 int,*

*@param2 char(5)*

)

RETURNS *@returntable* TABLE

(

*c1 int,*

*c2 char(5)*

)

AS

BEGIN

INSERT *@returntable*

SELECT *@param1, @param2*

RETURN

END;

GO

* UDF options :

ENCRYPTION

SCHEMABINDING

RETURNS NULL ON NULL INPUT / CALLED ON NULL INPUT

Test : p510

1 C

2 D

3 C

## XM11L

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/1