

Toutes les nouveautés de MS SQL Server 2005

par [SQLPro](#) ([autres articles](#)) ([Blog](#))

Date de publication : 2005

Dernière mise à jour :

Microsoft annonce la nouvelle moutûre de son SGBDR SQL Server 2005 pour le 7 novembre (2005 bien entendu !) à moins qu'une faille majeure de sécurité débusquée pendant les tests ne retarde son lancement. Profitons-en pour en découvrir toutes les nouvelles fonctionnalités...


Sachez dorénavant que Microsoft positionne SQL Server 2005 comme le champion absolu d'Internet et des solutions web, notamment avec du XML validé et indexé, et des web services intégrés. Mais aussi en champion de l'informatique décisionnelle avec la suite BI dev Studio, complétée par les modules SSIS (un véritable ETL remplaçant DTS) et Reporting Services l'outil de génération d'états... En attendant Maestro, la première pierre d'un édifice qui pourrait bien sonner le glas de Cognos ou Business Object !

I - Introduction.....	3
II - Améliorations apportées au langage SQL.....	3
II-A - Jointure externes (norme SQL 1992).....	3
II-B - Opérateur EXCEPT et INTERSECT (SQL:1992).....	3
II-C - Nouveau type XML (norme SQL:2003).....	4
II-D - Amélioration des types existants.....	4
II-E - Écriture des expressions de table (CTE : Common Table Expression, norme SQL:1999).....	5
II-F - Amélioration de l'intégrité référentielle (norme SQL:1992).....	6
II-F-1 - IR et SET NULL.....	6
II-F-2 - IR et SET DEFAULT.....	6
II-G - Fonctions de classement et d'énumération (norme SQL:2003).....	7
II-H - Opérateur PIVOT / UNPIVOT.....	8
II-I - Opérateur APPLY.....	10
II-J - Clause OUTPUT.....	11
II-K - Amélioration et extension de la clause TOP.....	12
II-L - Échantillon de table (TABLE SAMPLE).....	13
II-M - Schéma SQL (norme SQL:1992).....	13
II-N - Sécurité.....	13
II-O - Vues systèmes.....	14
II-P - Conclusion.....	14
III - Les nouveautés du langage Transact SQL.....	14
III-A - Terminaison des lignes.....	14
III-B - TABLE réparties.....	14
III-C - INDEX.....	15
III-D - Gestion des exceptions.....	16
III-E - Trigger de DDL.....	17
III-F - Niveau d'isolation SNAPSHOT.....	19
III-G - Base de données SNAPSHOT (cliché).....	19
III-H - Base de données miroir.....	19
III-I - Nouveaux paramètres de base de données.....	19
III-J - Plan de requête.....	20
III-K - Journal.....	20
III-L - Envoi de mails.....	20
III-M - Verrous mortels.....	21
III-N - Synonymes.....	23
III-O - Cryptage.....	24
III-P - Dépersonnalisation (Execute AS).....	24
III-Q - Conclusion.....	25
IV - Conclusion sur MS SQL Server 2005.....	25
V - Références.....	26

I - Introduction

Cet article est en plusieurs parties :

- la première partie traite des **nouveautés du langage SQL**
- la seconde partie traite des **nouveautés du langage Transact SQL**
- la troisième partie traite des extensions SQLCLR et XML du langage Transact SQL
- la quatrième partie présente les modules complémentaires de MS SQL Server 2005 tels que sqlcmd.exe (remplaçant osql et isql), SSIS (remplaçant DTS), BI dev studio (pour des bases OLAP), service broker (pour les Web Services), Notification Services (pour être tenu informé de changements d'état dans une base) ...

 *Les parties 3 et 4 sont en cours de réalisation. La partie 3 sera visible vers le 1er août et la 4 au plus tard le 1er septembre. Ici vous pourrez obtenir une version beta de MS SQL Server 2005. En fonction de votre plateforme (32 ou 64 bits) et du type d'OS (Windows 2003 server ou XP) choisissez la version Standard ou Entreprise pour un serveur ou bien Developer ou Workgroup pour un poste sous XP.*

II - Améliorations apportées au langage SQL

Voici une liste que j'espère suffisamment exhaustive des nouveautés de MS SQL Server 2005 et terme de "pur" langage SQL...

II-A - Jointure externes (norme SQL 1992)

Excellente nouvelle : les jointures externes avec une syntaxe propre à SQL server et rendant des résultats souvent faux, ne sont désormais plus acceptées nativement. Cette nouvelle ne va pas réjouir un certains nombre de développeurs et de DBA qui se la sont "coulé douce" depuis une décennie... Les jointures externe normatives sont dans la norme depuis 1992 et ont commencées à être supportées par SQL Server version 6.5. MS a mis en garde ses utilisateurs depuis la version 2000 qu'il était nécessaire d'utiliser les jointures normatives à base de LEFT, RIGHT ou FULL OUTER JOIN parce qu'elles ne seraient plus supportées dans les version futures...

Le message d'erreur est alors le suivant :

Msg 4147, Level 15, State 1, Line ...

The query uses non-ANSI outer join operators ("*=" or "=*"). To run this query without modification, please set the compatibility level for current database to 80 or lower, using stored procedure sp_dbcmptlevel. It is strongly recommended to rewrite the query using ANSI outer join operators (LEFT OUTER JOIN, RIGHT OUTER JOIN). In the future versions of SQL Server, non-ANSI join operators will not be supported even in backward-compatibility modes.

II-B - Opérateur EXCEPT et INTERSECT (SQL:1992)

Dorénavant, SQL Server se conforme à la norme et propose les opérateurs ensemblistes EXCEPT (différence) et INTERSECT (intersection). Cependant, et comme c'est le cas de l'actuel UNION, ces trois opérateurs n'implémentent pas le prédicat USING.

Exemple pour la différence :

Avec les tables :

```
CREATE TABLE MACHINE
(MAC_NOM VARCHAR(12))

CREATE TABLE OBJET
(OBJ_NOM VARCHAR(12))
```

Et les données :

```
INSERT INTO MACHINE VALUE('Moto')
INSERT INTO MACHINE VALUE('Perçeuuse')
INSERT INTO MACHINE VALUE('Avion')
INSERT INTO MACHINE VALUE('Ventilateur')
INSERT INTO MACHINE VALUE('Réveil')
```

Et les données :

```
INSERT INTO OBJET VALUE ('Moto')
INSERT INTO OBJET VALUE ('Assiette')
INSERT INTO OBJET VALUE ('Livre')
INSERT INTO OBJET VALUE ('Table')
INSERT INTO OBJET VALUE ('Perçuse')
```

Avant SQL Server 2005

```
SELECT OBJ_NOM
FROM T_OBJET
WHERE OBJ_NOM NOT IN (SELECT MAC_NOM
FROM T_MACHINE)
```

Ou bien

```
SELECT O.OBJ_NOM
FROM T_OBJET O
LEFT OUTER JOIN T_MACHINE M
ON O.OBJ_NOM = M.MAC_NOM
GROUP BY O.OBJ_NOM
HAVING COUNT(M.MAC_NOM) = 0
```

OBJ_NOM


```
-----
ASSIETTE
LIVRE
TABLE
```

Avec SQL Server 2005

```
SELECT OBJ_NOM
FROM T_OBJET
EXCEPT
SELECT MAC_NOM
FROM T_MACHINE
```

II-C - Nouveau type XML (norme SQL:2003)

SQL Server accepte dorénavant le type XML comme le prévoit la norme SQL:2003. Ce type SQL permet d'insérer une grappe XML valide ou un document XML validé par rapport à un schéma XSD. De plus des méthodes spécifiques ont été introduites et un document XML validé peut être indexé de manière interne (voir ci après).

 Les types DATE et TIME étaient prévus mais n'ont finalement pas été intégrés à cette version. Des problèmes de performances dans l'implémentation de ces types seraient à l'origine de la décision de MS de ne pas les intégrer à la version primale de MS SQL Server 2005


II-D - Amélioration des types existants

Les types VARCHAR, NVARCHAR et VARBINARY ont été améliorés et peuvent franchir la barre des 8000 octets. Pour cela, il faut indiquer une taille indéfinie à l'aide du marqueur "max".

Exemple

```
CREATE TABLE T_MAX
(MAX_VARCHAR VARCHAR(max),
MAX_NVARCHAR NVARCHAR(max),
MAX_VARBIN VARBINARY(max))

INSERT INTO T_MAX VALUES (REPLICATE('X', 10000), REPLICATE('Y', 100000),
CAST(REPLICATE('F', 1000000) AS VARBINARY(max)))
```

 Ils restent néanmoins limités à 2 Go de données.

! Prenez simplement conscience qu'un LIKE sur une colonne de type VARCHAR(max) contenant en moyenne 60 000 caractères dans une table de plusieurs dizaines de milliers de lignes, risque de prendre... un certain temps !

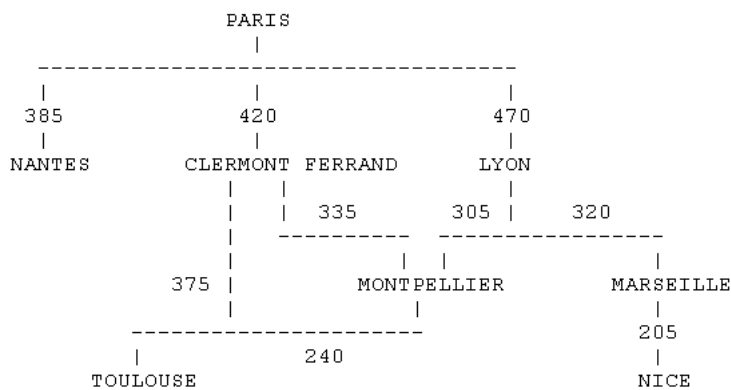
II-E - Écriture des expressions de table (CTE : Common Table Expression, norme SQL:1999)

Les expressions de tables permettent des écritures très synthétiques de requêtes SQL. Elles ont été introduite avec la norme SQL:1999. Leur intérêt principal est de permettre des expression de requêtes récursives. Extrait de mon livre "SQL" Pearson Education 2005, collection Synthex, co écrit avec Christian Soutou :

"Une expression de table consiste à exprimer une requête SELECT que l'on considérera comme une table dans la requête qui suivra cette expression. La création d'une vue répond à ce même principe tout en étant considéré comme un objet persistant de la base de données tandis que l'expression de table est créée dynamiquement et pour les besoins spécifique de la requête.

L'expression de table est utile pour simplifier certaines requêtes. Elle s'avère indispensable pour permettre un traitement récursif des données de la requête, parce qu'une corrélation est possible entre l'expression de table et la requête qui la construit."

Exemple : soit le réseau de transport suivant (théorie des graphes) ...



... composé de la table :

```
CREATE TABLE T_JOURNEY
(JNY_FROM_TOWN VARCHAR(32),
 JNY_TO_TOWN   VARCHAR(32),
 JNY_MILES     INTEGER)
```

... avec les données :

```
INSERT INTO T_JOURNEY VALUES ('PARIS', 'NANTES', 385)
INSERT INTO T_JOURNEY VALUES ('PARIS', 'CLERMONT-FERRAND', 420)
INSERT INTO T_JOURNEY VALUES ('PARIS', 'LYON', 470)
INSERT INTO T_JOURNEY VALUES ('CLERMONT-FERRAND', 'MONTPELLIER', 335)
INSERT INTO T_JOURNEY VALUES ('CLERMONT-FERRAND', 'TOULOUSE', 375)
INSERT INTO T_JOURNEY VALUES ('LYON', 'MONTPELLIER', 305)
INSERT INTO T_JOURNEY VALUES ('LYON', 'MARSEILLE', 320)
INSERT INTO T_JOURNEY VALUES ('MONTPELLIER', 'TOULOUSE', 240)
INSERT INTO T_JOURNEY VALUES ('MARSEILLE', 'NICE', 205)
```

Une question que l'on peut se poser est la suivante : quel est le plus court chemin pour aller de Paris à Toulouse ? Avant l'introduction du CTE et des requêtes récursives, il n'existait pas de solution sous forme d'une requête SQL à cette simple question. Désormais, la requête suivante répond à la question :


```
WITH
```

```

journey (DESTINATION, ETAPES, DISTANCE, CHEMIN)
AS
(SELECT DISTINCT JNY_FROM_TOWN, 0, 0, CAST('PARIS' AS VARCHAR(MAX))
FROM T_JOURNEY
WHERE JNY_FROM_TOWN = 'PARIS'
UNION ALL
SELECT JNY_TO_TOWN, departure.ETAPES + 1,
departure.DISTANCE + arrival.JNY_MILES,
departure.CHEMIN + ', ' + arrival.JNY_TO_TOWN
FROM T_JOURNEY AS arrival
INNER JOIN journey AS departure
ON departure.DESTINATION = arrival.JNY_FROM_TOWN),
short (DISTANCE)
AS
(SELECT MIN(DISTANCE)
FROM journey
WHERE DESTINATION = 'TOULOUSE')
SELECT *
FROM journey j
INNER JOIN short s
ON j.DISTANCE = s.DISTANCE
WHERE DESTINATION = 'TOULOUSE'

```

DESTINATION	ETAPES	DISTANCE	CHEMIN
TOULOUSE	2	795	PARIS, CLERMONT-FERRAND, TOULOUSE

 La clause **WITH** introduisant le **CTE** est utilisable aussi dans les ordres **DELETE**, **INSERT**, **UPDATE** et dans la construction des vues. Par exemple on peut supprimer tout un sous arbre partant d'un noeud dans une table présentant une hiérarchie en auto référence.

Vous pouvez lire [ici](#) un article très complet sur le sujet. Il doit paraître en français dans SQL Server magazine d'octobre 2005.

II-F - Amélioration de l'intégrité référentielle (norme SQL:1992)

Deux nouvelles manière de gérer l'intégrité référentielle ont été implémentées. Bien que datant de la norme SQL2 (1992), MS SQL Server était en retard à ce sujet.

II-F-1 - IR et SET NULL

Désormais en cas de modification de la valeur de la clef, comme en cas de suppression de la référence, les lignes filles peuvent voir leur clefs étrangères valuées à NULL.

II-F-2 - IR et SET DEFAULT

Désormais en cas de modification de la valeur de la clef, comme en cas de suppression de la référence, les lignes filles peuvent voir leur clefs étrangères valuées à la valeur par défaut définie dans l'expression de construction de la table. L'intérêt de ces deux nouvelles règles de gestion de l'intégrité référentielle réside dans l'obtention de performances par déport de l'effort de suppression comme par minimisation du verrouillage. En effet, avec un tel mécanisme, la suppression des lignes filles des tables référencées peut se faire dans un batch de nuit (ramasse miette), plutôt que par une cascade de "delete" à un moment de la journée ou la charge est maximale.

Exemple

```

CREATE TABLE T_CLIENT_CLI
(CLI_ID          INTEGER NOT NULL IDENTITY PRIMARY KEY,
 CLI_NOM        VARCHAR(32))

CREATE TABLE T_FACTURE_FAC
(FAC_ID         INTEGER NOT NULL IDENTITY PRIMARY KEY,
 FAC_DATE       DATETIME NOT NULL CURRENT_TIMESTAMP,
 FAC_MONTANT    FLOAT NOT NULL,

```

Exemple

```

CLI_ID          INTEGER DEFAULT 0
               FOREIGN KEY REFERENCES T_CLIENT_CLI (CLI_ID)
               ON UPDATE SET DEFAULT
               ON DELETE SET NULL)
    
```

II-G - Fonctions de classement et d'énumération (norme SQL:2003)

Certaines fonctions de classement et d'énumération, dites de fenêtrage, prévues par la norme SQL:2003 ont été ajoutées à SQL Server 2005 : RANK, DENSE_RANK et ROW_NUMBER. Microsoft a rajouté NTILE qui permet des regroupements par tranches énumérées par des valeurs discrètes.

Extrait de mon livre "SQL" Pearson Education 2005, collection Synthex, co écrit avec Christian Soutou :

Fonctions de fenêtrage et statistiques avancées (SQL:2003)

Les fonctions de "fenêtrage" s'appliquent au résultat de la requête et permettent par exemple la numérotation des lignes retournées ou l'établissement d'un rang. La syntaxe (simplifiée) est la suivante :

```

{ RANK ()
  | DENSE_RANK ()
  | PERCENT_RANK ()
  | CUME_DIST ()
  | ROW_NUMBER () } OVER <expression_specification>
    
```

RANK permet le classement absolu avec comptage des ex-aequo et DENSE_RANK sans le comptage des exaequo. PERCENT_RANK un classement en pourcentage. CUM_DIST la distribution cumulative et ROW_NUMBER numérote les lignes.


L'expression de spécification doit contenir l'ordre dans lequel les colonnes ou expressions doivent être triées pour que le classement ou la numérotation opère.

Exemple :

```

SELECT ROW_NUMBER() OVER(ORDER BY USR_ID) RNUM,
       USR_ID, USR_NOM, USR_PRENOM,
       RANK() OVER(ORDER BY USR_PRENOM) RANK,
       DENSE_RANK() OVER(ORDER BY USR_PRENOM) DENSE_RNK
FROM   T_UTILISATEUR_USR
    
```


RNUM	USR_ID	USR_NOM	USR_PRENOM	RANK	DENSE_RNK
18	28	BIDAL	Agnès	1	1
13	20	VALLADON	Antoine	2	2
8	11	MAILLANT	Catherine	3	3
11	16	DUPONT	Charles	4	4
16	25	DUPONT	Charles	4	4
1	1	DUPONT	François	6	5
10	15	CLERC	François	6	5
6	9	COWARD	John Leslie	8	6
3	4	MÜLLER	Marcel	9	7
5	7	LEROY	Olivier	10	8
12	18	LEROY	Émilie	11	9
2	2	DUVAL	Éric	12	10
7	10	PRUD'HOMME	Étienne	13	11
4	6	Martin	NULL	14	12
17	27	MAILLANT	NULL	14	12
9	14	CÉSARI	NULL	14	12
14	21	BIDAULT	NULL	14	12
15	22	HERMEX	NULL	14	12

 Microsoft SQL Server 2005 n'implémente pas les fonctions normatives PERCENT_RANK et CUM_DIST.

Quant à la fonction NTILE, spécifique à SQL Server 2005, elle permet de regrouper par tranches contenant un nombre en principes égaux de lignes les valeurs des colonnes visées. Cela peut être utile pour classer par sous groupe les éléments d'un ensemble. Par exemple, dans une classe de 16 élèves on peut créer trois groupes de travail pour les cours de gymnastique. Les forts, les moins forts et les faibles. Dans ce cas la fonction NTILE placera les six premiers des épreuves de sélection sportive dans le groupe 1, puis six en suivant dans le groupe 2 et enfin, 4 dans le groupe 3. L'expression d'une telle requête pourrait prendre la forme :

```
SELECT ELV_NOM,
        NTILE(3) OVER(ORDER BY ELV_NOTE_GYM) AS ELV_GROUPE,
        ELV_NOTE_GYM
FROM T_ELEVE_ELV
```

ELV_NOM	ELV_GROUPE	ELV_NOTE_GYM
DUPONT	1	18
DUVAL	1	18
DUHAMEL	1	16
DUBOIS	1	15
DULAC	1	14,5
DUPOND	1	14
LAPORTE	2	14
LESTER	2	14
LAMBERT	2	13,5
LOUVIER	2	13,5
LAMAR	2	13
LEGRAND	2	12
MONNIER	3	12
MOHAMED	3	10
MICHAUD	3	9,5
MACHET	3	8

 ces fonctions de classement sont des fonctions dites de "fenêtrage". Elle ne s'appliquent qu'aux lignes devant être affichées et ne peuvent ni figurer dans une sous requête classique, ni dans une clause WHERE, HAVING, etc... parce qu'elles s'appliquent après que les lignes du résultat aient été construites. Ainsi il n'est pas possible d'utiliser la fonction ROW_NUMBER ou NTILE pour faire directement de la pagination. En revanche, une telle requête peut être encapsulée en tant que sous requête table dans la clause FROM et servir à différents usages.

Voici un exemple, de pagination dans un jeu de résultats avec deux méthodes différentes.

1) Pagination par bloc de ligne contrôlé (les lignes retournées sont contrôlées dans le filtre WHERE) :

```
SELECT *
FROM (SELECT ROW_NUMBER() OVER(ORDER BY USR_ID) RNUM,
            USR_ID, USR_NOM, USR_PRENOM,
            FROM T_UTILISATEUR_USR) T
WHERE RNUM BETWEEN 10 AND 19
```

Ici on retourne les lignes numérotées 10 à 19.

2) Pagination par nombre de page (le nombre de page est fixé, le nombre de ligne inconnu, la page retournée est contrôlée dans le filtre WHERE) :

```
SELECT *
FROM (SELECT NTILE(25) OVER(ORDER BY USR_ID) NTL,
            USR_ID, USR_NOM, USR_PRENOM,
            FROM T_UTILISATEUR_USR) T
WHERE NTL = 7
```

Ici on fixe le nombre de page à 25 et l'on retourne la page 7

Une autre construction plus élégante est possible via la CTE.

II-H - Opérateur PIVOT / UNPIVOT

L'horrible, mais traditionnel opérateur PIVOT, bien qu'absent de la norme SQL fait son apparition dans SQL Server 2005 afin de permettre de réaliser des tableaux croisés. Il s'agit simplement d'une astuce cosmétique pour présenter des données avec de nouvelles colonnes dont les noms sont extraits d'une des colonnes de la requête.

Exemple :

```

CREATE TABLE T_VENTE_VTE
(VTE_ID          INT,
 VTE_PERIODE     VARCHAR(8),
 VTE_ZONE        VARCHAR(6),
 VTE_CA          FLOAT)

INSERT INTO T_VENTE_VTE VALUES (1, 'TR 1', 'Sud', 232534.34)
INSERT INTO T_VENTE_VTE VALUES (2, 'TR 2', 'Sud', 565537.65)
INSERT INTO T_VENTE_VTE VALUES (4, 'TR 3', 'Sud', 254537.89)
INSERT INTO T_VENTE_VTE VALUES (5, 'TR 4', 'Sud', 345564.98)
INSERT INTO T_VENTE_VTE VALUES (6, 'TR 1', 'Nord', 234507.67)
INSERT INTO T_VENTE_VTE VALUES (7, 'TR 3', 'Nord', 787537.88)
INSERT INTO T_VENTE_VTE VALUES (8, 'TR 4', 'Nord', 675455.62)


-- Avant SQL Server 2005 :
SELECT VTE_ZONE, SUM(VTE_CA) AS TRIMESTRE_1,
       (SELECT SUM(VTE_CA)
        FROM T_VENTE_VTE
        WHERE VTE_PERIODE = 'TR 2'
              AND VTE_ZONE = VTE.VTE_ZONE) AS TRIMESTRE_2,
       (SELECT SUM(VTE_CA)
        FROM T_VENTE_VTE
        WHERE VTE_PERIODE = 'TR 3'
              AND VTE_ZONE = VTE.VTE_ZONE) AS TRIMESTRE_3,
       (SELECT SUM(VTE_CA)
        FROM T_VENTE_VTE
        WHERE VTE_PERIODE = 'TR 4'
              AND VTE_ZONE = VTE.VTE_ZONE) AS TRIMESTRE_3
FROM T_VENTE_VTE VTE
WHERE VTE_PERIODE = 'TR 1'
GROUP BY VTE_ZONE

-- Avec SQL Server 2005 :
SELECT VTE_ZONE, SUM([TR 1]) AS TRIMESTRE_1, SUM([TR 2]) AS TRIMESTRE_2,
       SUM([TR 3]) AS TRIMESTRE_3, SUM([TR 4]) AS TRIMESTRE_4
FROM T_VENTE_VTE
PIVOT (SUM(VTE_CA) FOR VTE_PERIODE IN ([TR 1], [TR 2], [TR 3], [TR 4]))
      AS TRIMESTRE
GROUP BY VTE_ZONE
    
```

VTE_ZONE	TRIMESTRE_1	TRIMESTRE_2	TRIMESTRE_3	TRIMESTRE_4
Nord	234507,67	NULL	787537,88	675455,62
Sud	232534,34	565537,65	254537,89	345564,98

```

-- les mêmes résultats sans la cosmétique du tableau croisé :
SELECT VTE_ZONE, VTE_PERIODE, SUM(VTE_CA) AS CA
FROM T_VENTE_VTE VTE
GROUP BY VTE_ZONE, VTE_PERIODE
    
```

 **Si cette technique permet de réconcilier les aficionados d'Access et des tableaux croisés avec SQL Server, tant mieux. Mais ce "truc" possède de très nombreux inconvénients majeurs :**

- 1) Aucune conformité aux normes SQL et franchement pas standard : la transformation de valeurs en nom de colonnes est une absurdité et le recours aux crochets pour nommer les colonnes provenant des valeurs n'obéit à aucune logique, ni aucun standard si ce n'est les spécificités de MS SQL Server.
- 2) Même si le plan de requête paraît plus simple, il n'est pas certain que l'effort en terme d'E/S soit moindre. En effet l'utilisation d'un tel opérateur empêche en principe l'utilisation des index.

3) Toute opération cosmétique grève énormément les performances d'un serveur SQL parce qu'il n'a pas été prévu pour cela, alors qu'un outil de présentation spécialement conçu à cet effet pourra donner le même rendu en minimisant les ressources.

4) La clause IN de l'opérateur PIVOT n'a pas été rendue générique. En l'occurrence l'écriture très tentant d'une requête du genre :

```
PIVOT (SUM(VTE_CA) FOR VTE_PERIODE IN (VTE_PERIODE) AS TRIMESTRE
```

ou encore :

```
PIVOT (SUM(VTE_CA) FOR VTE_PERIODE
      IN (SELECT DISTINCT VTE_PERIODE FROM T_VENTE_VTE) AS TRIMESTRE
```

n'est pas possible. Il faut spécifier "en dur" et connaître d'avance les colonnes !

En conclusion : les opérateurs PIVOT / UNPIVOT revêtent peu d'intérêt et sont à éviter systématiquement pour qui veut des performances.

UNPIVOT est l'opération inverse de PIVOT, mais ne peut reproduire exactement les données initiales du fait du traitement spécifiques des absences de valeurs (marqueurs NULL).

Pour compléter votre information, l'article de [Reanaud Harduin sur le sujet](#).

II-I - Opérateur APPLY

L'opérateur APPLY a pour but d'appliquer une jointure à un ensemble de ligne extrait d'une colonne d'une table et sa table hôte. Mais, me direz-vous, est-il possible de mettre des lignes dans une colonne de table ? Oui, si l'on considère qu'une fonction table peut être appliquée pour une colonne, ou qu'une colonne peut contenir du xml dont l'extraction d'un noeud peut retourner un ensemble de lignes...

Voici un exemple, complet...

La fonction table suivante retourne une table constituée des 7 jours de la semaine contenant la valeur du paramètre DATETIME qu'on lui passe :

```
CREATE FUNCTION F_WEEKDAYTABLE (@A_DATE DATETIME)
RETURNS @WEEKDAYTABLE TABLE ("DATE" DATETIME primary key,
                              JOUR    VARCHAR(8))
BEGIN
-- obtention de la date avec heure 0
SET @A_DATE = CAST(FLOOR(CAST(@A_DATE AS FLOAT)) AS DATETIME);
-- recherche du lundi
WHILE DATEPART(WEEKDAY, @A_DATE) <> 1
SET @A_DATE = @A_DATE -1
-- insertion des jours de la semaine dans la table
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE, 'Lundi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 1, 'Mardi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 2, 'Mercredi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 3, 'Jeudi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 4, 'Vendredi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 5, 'Samedi')
INSERT INTO @WEEKDAYTABLE VALUES (@A_DATE + 6, 'Dimanche')
-- retour
RETURN
END

-- la table suivante contient des factures :
CREATE TABLE T_FACTURE_FCT
(FCT_ID      INTEGER NOT NULL PRIMARY KEY,
 FCT_DATE    DATETIME NOT NULL,
 CLI_ID      INTEGER NOT NULL)

INSERT INTO T_FACTURE_FCT VALUES (145, '20050718', 33)
INSERT INTO T_FACTURE_FCT VALUES (178, '20050720', 21)
INSERT INTO T_FACTURE_FCT VALUES (213, '20050722', 47)
```

FCT_ID	FCT_DATE	CLI_ID
145	2005-07-18 00:00:00.000	33
178	2005-07-20 00:00:00.000	21
213	2005-07-22 00:00:00.000	47

Les factures doivent être envoyées le jeudi qui suit la date de la facture. Comment exprimer cela à l'aide des éléments ci dessous et obtenir :

FCT_ID	FCT_DATE	CLI_ID	FCT_DATE_ENVOI
145	2005-07-18 00:00:00.000	33	2005-07-21 00:00:00.000
178	2005-07-20 00:00:00.000	21	2005-07-21 00:00:00.000
213	2005-07-22 00:00:00.000	47	2005-07-28 00:00:00.000

Une première tentative :

```
SELECT *
FROM T_FACTURE_FCT F
CROSS JOIN dbo.F_WEEKDAYTABLE (F.FCT_DATE) W
WHERE W.JOUR = 'Jeudi'
```

Se solde par une erreur à la compilation :

Serveur : Msg 170, Niveau 15, État 1, Ligne 3

Ligne 3 : syntaxe incorrecte vers '!'.

En effet on ne peut pas exprimer une colonne d'une table de la clause FROM en paramètre dans une fonction contenue dans cette même clause FROM. Ce qui était impossible sous SQL Server 2000 devient possible en version 2005 : la solution passe par l'un des deux nouveaux opérateurs de jointure CROSS APPLY ou OUTER APPLY...

```
SELECT FCT_ID, FCT_DATE, CLI_ID,
CASE
WHEN FCT_DATE > W.DATE THEN WW.DATE
ELSE W.DATE
END AS DATE_ENVOI
FROM T_FACTURE_FCT F
CROSS APPLY dbo.F_WEEKDAYTABLE (F.FCT_DATE) W
CROSS APPLY dbo.F_WEEKDAYTABLE (F.FCT_DATE + 7) WW
WHERE W.JOUR = 'Jeudi'
AND WW.JOUR = 'Jeudi'
```

FCT_ID	FCT_DATE	CLI_ID	DATE_ENVOI
145	2005-07-18 00:00:00.000	33	2005-07-20 00:00:00.000
178	2005-07-20 00:00:00.000	21	2005-07-20 00:00:00.000
213	2005-07-22 00:00:00.000	47	2005-07-27 00:00:00.000

MS SQL Server dispose donc de deux opérateurs APPLY : CROSS APPLY pour un produit cartésien et OUTER APPLY pour une jointure externe.

II-J - Clause OUTPUT

Voici une clause intéressante pour les ordres SQL de mise à jour. Elle permet de tout connaître de ce qui s'est passé dans un tel ordre (INSERT, UPDATE, DELETE). Son principe consiste à répercuter les informations d'ajouts, modifications ou suppressions dans une table de votre choix. Voici un exemple d'utilisation d'une telle clause.


```
-- création d'une table de trace pour pister les modifications
CREATE TABLE T_TRACE_TRC
(TRC_ID INTEGER IDENTITY NOT NULL PRIMARY KEY,
TRC_DATEHEURE DATETIME DEFAULT CURRENT_TIMESTAMP,
TRC_TABLE NVARCHAR(128),
TRC_CLEF INTEGER,
TRC_DATA NVARCHAR(256),
TRC_HOST_NAME NVARCHAR(128),
TRC_PROGRAM_NAME NVARCHAR(128),
TRC_NT_USER_NAME NVARCHAR(128),
TRC_NET_ADDRESS NCHAR(12),
```

```

TRC_LOGIN_NAME          NVARCHAR(128)

-- ordre DELETE avec clause OUTPUT :
DELETE FROM T_CHAMBRE_CHB
OUTPUT CURRENT_TIMESTAMP,
       'T_CHAMBRE_CHB',
       deleted.CHB_ID,
       'CHB_NUM = ' + CAST(deleted.CHB_NUM AS VARCHAR(16)) +
       ', CHB_COUCHAGE = '+CAST(deleted.CHB_COUCHAGE AS VARCHAR(16)),
       p.host_name,
       p.program_name,
       p.nt_user_name,
       p.net_address,
       p.login_name
INTO T_TRACE_TRC
FROM   T_CHAMBRE_CHB CHB
CROSS JOIN master.dbo.sysprocesses p
WHERE  CHB.CHB_COUCHAGE >= 5
AND    p.spid = @@spid
    
```

Dans cet exemple nous faisons une jointure croisée avec la tables des processus pour le processus en cours afin de rapatrier les informations de l'utilisateur, du programme et du noeud réseau sur lequel il travaille afin d'alimenter la table de trace avec ces informations, comme avec les informations concernant les éléments supprimés. Notez la présence de la pseudo table deleted qui contient les lignes en cours de suppression, comme dans le cadre d'un trigger. De même la table inserted est disponible dans la clause OUTPUT.

 Dans sa première version, la clause OUTPUT ne pouvait alimenter qu'une variable de type table devant être préalablement déclarée. Il a été annoncé que cette clause OUTPUT pourrait alimenter une table persistante quelconque. C'est ce que nous avons fait dans notre exemple. Néanmoins nous n'avons pas constaté dans le bêta 3 cette évolution.

II-K - Amélioration et extension de la clause TOP

Une des critiques souvent faite sur cette clause qui permet de limiter le nombre de lignes retournées est qu'elle n'était pas paramétrable. C'est désormais chose faite. On peut mettre dans TOP une variable comme une sous requête. Voici un exemple qui permet de retourner la moitié des lignes d'une table :

```

SELECT TOP (SELECT COUNT(*) / 2
            FROM T_CHAMBRE)
        *
FROM   T_CHAMBRE
    
```


Il est maintenant possible d'utiliser la clause TOP dans les autres ordres SQL du DDL : INSERT, UPDATE, DELETE. Cela peut être très intéressant pour découper en plusieurs lots une requête qui écrit des données et donc minimiser le volume des transactions.

Exemple :

```

WHILE EXISTS (SELECT *
              FROM   T_CLIENT
              WHERE  CLI_NOM <> UPPER(CLI_NOM))
UPDATE T_CLIENT
TOP    400
SET    CLI_NOM = UPPER(CLI_NOM)
    
```

Dans cette requête, on met en majuscules le nom des clients par paquet de 400 tant qu'il existe des noms de client qui ne l'ont pas encore été.

 Autrefois, il fallait recourir pour ce faire à l'indicateur ROWCOUNT. Microsoft annonce que celle-ci ne sera plus supportée dans les versions futures de SQL Server.

II-L - Échantillon de table (TABLE SAMPLE)

Cette nouvelle clause permet de retourner un échantillon de lignes de la table plutôt que de retourner toutes les lignes. Cette clause complète un peu la clause TOP, avec un avantage, les lignes sont retournées dans des pages de la table prises au hasard, alors que TOP implique un classement ce qui implique donc échantillon organisé donc peu représentatif.

Il est même prévu que la méthode d'échantillonnage puisse être choisie, voire d'implémenter sa propre méthode.

La syntaxe est la suivante :

```
TABLESAMPLE [SYSTEM] (nombre [ PERCENT | ROWS ] ) [ REPEATABLE (graine) ]
```


Exemple :

```
SELECT *
FROM T_FACTURE TABLESAMPLE (2 PERCENT)
```

Dans le cas où l'on désire obtenir toujours le même jeu de lignes dans l'échantillon, on peut proposer une valeur numérique arbitraire pour la valeur graine. Dès lors, avec l'utilisation de cette même valeur, l'échantillon de pages retournées sera le même à chaque exécution, à condition que les pages n'aient pas été réorganisées (DBCC INDEXDEFRAG par exemple).

Exemple :

```
SELECT *
FROM T_FACTURE TABLESAMPLE (2 PERCENT) REPEATABLE 12345
```

 Comme TABLESAMPLE fonctionne sur des pages et non des lignes, le nombre de ligne est approximatif et dépend du nombre de ligne par page. Ceci peut être pénalisant dans de petite tables ou TABLESAMPLE peut ne rien renvoyer dans le cas d'un 40% par exemple si la totalité des lignes est contenue dans une page. Conséquence : TABLESAMPLE n'a d'intérêt que pour des tables volumineuses (plusieurs centaines de pages...).

II-M - Schéma SQL (norme SQL:1992)

La notion de schéma SQL a été améliorée (elle se trouve maintenant proche de la norme SQL2) et il est désormais possible de créer dans une même base différents schémas sans passer obligatoirement par un nouvel utilisateur. Cela revient à dire qu'un objet n'appartient plus directement à un utilisateur, mais à un schéma. En outre différents utilisateurs peuvent ajouter, modifier ou retirer des objets du schéma pourvu qu'ils en aient acquis les privilèges.

II-N - Sécurité

La sécurité a été grandement améliorée : des pseudo ordre SQL sont disponibles là où des procédures complexes et absconces devaient être employées. Ainsi on trouve les ordres Transact SQL :

```
CREATE USER ... WITH DEFAULT_SCHEMA ...
CREATE LOGIN ...
GRANT ... ON SCHEMA ... TO ...
```

Comme nous l'avons déjà dit, la notion d'utilisateur SQL devient indépendante du schéma.

Un accès au serveur peut être formé avec un certificat (CERTIFICATE ou CREDENTIAL) ou une clef de chiffrement asymétrique. Cette technique est notamment requise pour authentifier certains processus notamment dans le cadre de l'utilisation des web services intégrés à MS SQL Server et par conséquent faire dialoguer des Serveurs SQL à travers le Web

Enfin, il est possible de caler la politique de gestion des mots de passe ("password policies") sur celle de Windows Server.,


Notons que la gestion des privilèges (GRANT, REVOKE) a été entièrement revue afin de gérer tous les nouveaux objets de SQL Server 2005. Ce qui existait sous SQL Server 2000 reste, bien entendu compatible. Les privilèges sont imbriqués de manière hiérarchique.

II-O - Vues systèmes

Microsoft utilise les nouvelles possibilités de schéma au sein des bases de données de SQL Server 2005 en ajoutant systématiquement à toute base de données créée un schéma de nom "sys" regroupant les vues systèmes interne à SQL Server.

Par exemple, la vue `sys.database_files` remplace avantageusement la table `sysfiles` :

- elle contient plus d'information que `sysfiles`
- les informations contenues sont plus claires
- les vues sont plus facile à sécuriser

 *Si vous devez recourir aux informations systèmes, préférez dans l'ordre :*

- 1 les vues normatives (SQL2) d'information de schéma (`INFORMATION_SCHEMA.TABLE` par exemple)
- 2 les vues systèmes (`sys.objects` par exemple)
- 3 les procédures spécialisées (`sp_help` par exemple) et en dernier recours
- 4 les tables systèmes (`sysobjects` par exemple)

En effet, seules les vues et les procédures sont compatibles d'une version à l'autre et les vues normatives sont à préférer pour des raisons de compatibilité.

II-P - Conclusion

Même s'il reste des efforts à faire du côté de SQL, tel que l'implémentation des types `DATE` et `TIME` (possible avec CLR), du type `ROW` ou `ARRAY` (ces derniers étant simulable par du xml) et qu'il manque encore certaines constructions SQL telle que le `Row Value Constructor`, le concept d'assertion ou encore les prédicats `UNIQUE`, `DISTINCT` ou `MATCH`, force est de constater que le SQL de MS SQL Server 2005 est d'un niveau élevé et largement suffisant pour la très grande majorité des développements.

III - Les nouveautés du langage Transact SQL

Voici une liste que j'espère suffisamment exhaustive des nouveautés de MS SQL Server 2005 en terme de langage Transact SQL...

III-A - Terminaison des lignes

Désormais il faut terminer toutes les lignes de code par le caractère point-virgule (;). Ceci est important car certaines syntaxes risquent de ne pas fonctionner correctement dans les procédures, les fonctions, les triggers ou tout simplement dans les différents ordres d'un fichier de "batch".

C'est encore une recommandation faible de MS, mais il y a des exemple déjà connus où cette syntaxe est requise, notamment avant le lancement de l'ordre `SEND (Service Broker)`.

III-B - TABLE réparties

Sous SQL Server 2000 ont été introduites les vues partitionnées (indexables).

SQL Server 2005 propose dorénavant les tables réparties (partitionnées étant peu français...). La technique consiste à diviser la table en plusieurs partitions logiques (et si possible physique, par exemple sur des grappes RAID distinctes, c'est tout l'intérêt de cette affaire) à l'aide d'une règle de partitionnement introduite par la commande `CREATE PARTITION...`

Voici, par exemple comment procéder pour équi-répartir une table contenant des noms de personnes à l'aide d'une classification de type `CUTTER-SANBORN` :

1) On crée la fonction de répartition

```
CREATE PARTITION FUNCTION PF_CUTTER (CHAR(25))
AS RANGE LEFT FOR VALUE ('COSTAZ', 'HOENNER', 'PANIER')
```

2) On crée la répartition basée sur la fonction établie précédemment

```
CREATE PARTITION SCHEMA PS_CUTTER
AS PARTITION PF_CUTTER
TO (FileGroup1, FileGroup2, FileGroup3, FileGroup4)
```

Notez que, comme la fonction de répartition compte trois valeurs pivot, il y aura quatre espace de stockage. C'est le vieux problème des intervalles...


Bien entendu, FileGroup1 ... FileGroup4 sont des groupes de fichiers préalablement créés sur des ressources de stockage distinctes.

3) Désormais, nous pouvons créer notre table partitionnée

```
CREATE TABLE T_EMPLOYEE_EMP
(EMP_ID                INTEGER NOT NULL,
 EMP_MATRICULE         CHAR(8) NOT NULL,
 EMP_NOM               CHAR(25),
 ...
) ON PS_CUTTER (EMP_NOM)
```

C'est la colonne EMP_NOM qui est pris en compte par le schéma de répartition avec sa fonction associée.

Désormais les lignes contenant les noms des employés depuis A... jusqu'à COSTAZ inclus (ordre alphabétique, et seront stockées dans le groupe de fichier FileGroup1, celles contenant les noms de COSTAZ exclu à HOENNER inclus dans FileGroup2 et ainsi de suite...

 *Je n'ai pas encore fait de tests pour savoir si la fonction de partitionnement est sensible à la collation ou non, notamment pour les problématiques de casse ou d'accents et autres caractères diacritiques. Ce test est cependant facile, car il est possible de savoir sur quelle partition se trouvent vos données...*

La l'opérateur \$PARTITION permet de récupérer le numéro de la partition pour une valeur spécifique. Cet opérateur doit être appliqué à la fonction de répartition. Il s'utilise comme ceci :

```
SELECT EMP_NOM, $PARTITION.PF_CUTTER(EMP_NOM)
FROM T_EMPLOYEE_EMP
```

Mieux, on peut compter le nombre de lignes de chacune des partitions pour voir si la répartition est équitable (c'est le but des tables de CUTTER-SANBORN !) :

```
SELECT $PARTITION.PF_CUTTER(EMP_NOM) AS PARTITION_NUM,
COUNT(*) AS NOMBRE_LIGNE
FROM T_EMPLOYEE_EMP
GROUP BY EMP_NOM
```

III-C - INDEX

On peut désormais contrôler le verrouillage des index avec les options ALLOW_ROW_LOCK et ALLOW_PAGE_LOCK disponibles dans les ordres CREATE et ALTER.

On peut contrôler aussi le parallélisme d'accès sur un index avec l'option MAXDOP (n)

On peut aussi désactiver un index avec ALTER INDEX ... DISABLE.

Comme une table, un index peut aussi être partitionné lors de sa création. Cela permet notamment de synchroniser les partitions aux niveau des données et des index.

ONLINE ON / OFF permet à un index d'être utilisé ou non pendant les opérations de construction ou de modification d'index.

Certaines opérations relativement pénible qui devaient être exécutées avec le DBCC sont désormais disponible via ALTER INDEX. Il s'agit de :

REBUILD, remplace le DBCC REINDEX

REORGANIZE remplace le DBCC INDEXDEFRAG

Mais la grande nouveauté consiste dans le fait que l'on puisse à ajouter des colonnes d'information non indexées dans un index, afin d'assurer la couverture de certaines opérations de récupération de l'information dans les requêtes afin d'éviter de recourir à des lectures multiples d'index. Cela se fait avec l'option INCLUDE.

Exemple :

Soit la table :

```
CREATE TABLE T_EMPLOYEE_EMP
(EMP_ID          INTEGER NOT NULL PRIMARY KEY,
 EMP_MATRICULE   CHAR(8) NOT NULL,
 EMP_TITRE       VARCHAR(12),
 EMP_NOM         CHAR(25),
 EMP_PRENOM      VARCHAR(16),
 EMP_DATE_NAISSANCE DATETIME,
 EMP_SALAIRE     FLOAT
)
```

SQL Server a automatiquement créé un index CLUSTER du fait de la présence de la clef primaire. Cependant, la requête suivante :

```
SELECT EMP_ID, EMP_MATRICULE, EMP_NOM
FROM T_EMPLOYEE_EMP
WHERE EMP_MATRICULE = '12345678'
```

Fera un balayage de la table à la recherche pour rechercher la ligne.

Mieux, en ajoutant un index sur EMP_MATRICULE de la sorte :

```
CREATE INDEX IX_EMP_MATRICULE ON T_EMPLOYEE_EMP (EMP_MATRICULE)
```

Le plan de requête cherchera dans l'index IX_EMP_MATRICULE la valeur '12345678', puis ayant obtenu l'identifiant, le recherchera dans l'index CLUSTER afin de reprendre l'information concernant le nom.

Dans un tel cas, la construction d'un index comme :

```
CREATE INDEX IX_EMP_MATRICULE_IN_NOM_OUT
ON T_EMPLOYEE_EMP (EMP_MATRICULE)
INCLUDE (EMP_NOM)
```

Permet de s'éviter la double lecture de l'index puisque toutes les informations requise pour alimenter les données à renvoyées sont présente dans l'index IX_EMP_MATRICULE_IN_NOM_OUT. Dans un tel cas, le temps de traitement est divisé par deux. Le seul inconvénient est un temps d'insertion et de modification des lignes plus grand du fait de l'effort plu important à réaliser afin d'alimenter le redondance.

III-D - Gestion des exceptions

S'il y avait bien un point à rectifier dans le Transact SQL, c'est bien celui là : la gestion des erreurs, et plus généralement des exceptions. L'écriture avant SQL Server 2005 était proche de celle du Cobol ! En effet, voici comment il fallait piloter une transaction composée de trois requêtes de mise à jour avec ma version 2000 :

```
BEGIN TRANSACTION

DELETE FROM ...
IF @@ERROR <> 0
    GOTO LBL_ROLLBACK_ON_ERROR
INSERT INTO ...
IF @@ERROR <> 0
    GOTO LBL_ROLLBACK_ON_ERROR
UPDATE ...
```

```

IF @@ERROR <> 0
    GOTO LBL_ROLLBACK_ON_ERROR
COMMIT TRANSACTION
RETURN

LBL_ROLLBACK_ON_ERROR:
ROLLBACK
    
```

Il n'y avait d'ailleurs pas moyen d'obtenir grand chose sur le pourquoi du comment de l'erreur, tout étant dans une unique information : le numéro de l'erreur !

Désormais, comme dans tout langage moderne, Transact SQL dispose avec SQL Server 2005 d'une structure de bloc de capture d'erreur. Voici la même procédure que ci dessus, avec l'encapsulation de la gestion d'exception :

```

BEGIN TRY
    BEGIN TRANSACTION
        DELETE FROM ...
        INSERT INTO ...
        UPDATE ...
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
END CATCH
    
```


Mais pour être parfait le code doit être un peu amélioré. En effet, cette version capture toutes les erreurs y compris les plus insignifiantes. pour ne plus s'occuper que des erreurs sur les phases de manipulations de données, voici comment procéder :

```

BEGIN TRY
    BEGIN TRANSACTION
        ...
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    IF XACT_STATE() = -1
        ROLLBACK TRANSACTION
    IF XACT_STATE() = 1
        COMMIT TRANSACTION
END CATCH
    
```

MS SQL Server 2005 introduit la fonction XACT_STATE capable de déterminer si la transaction est active et validable (1), active et en erreur (-1) ou bien inactive (0, transaction terminée et déjà validée ou pas de transaction du tout). En plus de cette fonction, certaines fonctions supplémentaires permettent d'obtenir des informations sur l'erreur génératrice de la levée d'exception :

ERROR_NUMBER	le numéro de l'erreur (vue sys.messages dans master).
ERROR_SEVERITY	le niveau de sévérité
ERROR_STATE	le niveau d'état de l'erreur
ERROR_PROCEDURE	le nom de la procédure stockée ou du trigger dans laquelle l'exception s'est produite
ERROR_LINE	le numéo de la ligne où s'est produite l'erreur
ERROR_MESSAGE	le message de l'erreur avec les emplacements de paramètres

 *L'utilisation des blocs TRY/CATCH nécessite le positionnement du flag XACT_ABORT à ON*

III-E - Trigger de DDL

Voici une idée quelque peu reprise d'Oracle : pourquoi ne pas mettre en oeuvre des triggers sur la création, la suppression ou la modification des objets d'une base, voir de lancer du code lors de la création ou la suppression d'une base de données ?

C'est ce que propose désormais MS SQL Server 2005 à travers les triggersd DDL (rapellons pour ceux qui ne sont pas familié avec le langage SQL que DDL signifie Data Definition Language, et c'est la partie du SQL qui s'occupe de créer, modifier ou supprimer les objets à l'aide des ordres CREATE, ALTER et DROP).

La syntaxe d'un trigger est la suivante :

```
CREATE TRIGGER nom
ON { DATABASE | ALL SERVER }
FOR <événement>
AS
...
```

Les événements capturables au niveau base de données sont les suivants :

CREATE_, ALTER_, DROP_	APPLICATION_ROLE, ASSEMBLY, CERTIFICATE, CONTRACT, FUNCTION, INDEX, MESSAGE_TYPE, PARTITION_FUNCTION, PARTITION_SCHEME, PROCEDURE, QUEUE, REMOTE_SERVICE_BINDING, ROLE, ROUTE, SCHEMA, SERVICE, TABLE, TRIGGER, USER, VIEW, XML_SCHEMA_COLLECTION
CREATE_, DROP_	EVENT_NOTIFICATION, SYNONYM, TYPE, STATISTICS
(autres)	ALTER_AUTHORIZATION_DATABASE, GRANT_DATABASE, DENY_DATABASE, REVOKE_DATABASE, UPDATE_STATISTICS

Les événements capturables au niveau serveur sont les suivants :

```
ALTER_AUTHORIZATION_SERVER
CREATE_ENDPOINT
DROP_ENDPOINT
CREATE_LOGIN
ALTER_LOGIN
DROP_LOGIN
GRANT_SERVER
DENY_SERVER
REVOKE_SERVER
```

En sus, SQL Server 2005 fournit un "paquet" informatif sur l'événement pisté par le trigger, et accessible via la fonction EVENTDATA qui retourne un document XML. Bien entendu les attributs disponibles dans ce document diffèrent en fonction de la nature de l'événement suivi.

Voici un exemple d'utilisation d'un tel trigger qui alerte par mail l'administrateur de bases de données de la création d'une nouvelle table dans la base et lui envoie le texte de l'ordre SQL de création de cette table. Notez comment a été extrait du document XML l'information qui nous intéressait :

```
CREATE TRIGGER E_ALERTER_CREATION_TABLE
ON DATABASE
FOR CREATE_TABLE
AS
DECLARE @SQLORDER VARCHAR(max)
SELECT @SQLORDER = EVENTDATA().value (' (/EVENT_INSTANCE/TSQLCommand/CommandText) [1] ',
'nvarchar(max)')
EXEC xp_sendmail @recipients = 'dba@myOrganisation.com',
@subject = 'Table créée dans la base ' + DB_NAME(),
@message = @SQLORDER
```

EVENTDATA() retourne le flux XML dont on extrait la valeur de l'élément "/EVENT_INSTANCE/TSQLCommand/CommandText)[1]" à l'aide de la fonction XQuery value.

III-F - Niveau d'isolation SNAPSHOT

Un nouveau niveau d'isolation, hors norme, de type SNAPSHOT a été ajouté. Il permet de réaliser ses transactions sur un cliché (un instantané des objets scrutés). Il est basé sur un verouillage optimiste et évite donc un tout verouillage en lecture. Voici un niveau d'isolation idéal pour assurer des lectures complexes de données avec de forts volumes (calculs statistiques massifs sur des données en production par exemple)

En complément, l'article de [Ravindra Okade sur le sujet](#) .

III-G - Base de données SNAPSHOT (cliché)

Il est désormais possible de créer une base de données de type SNAPSHOT et de travailler dessus. Une base de données SNAPSHOT n'est autre qu'un cliché à un instant T d'une quelconque base de données de SQL Server 2005. Le mécanisme de SNAPSHOT est extrêmement rapide puisqu'au départ, aucune pas n'est copiée. Seules les pages qui vont être modifiées dans le futur et dans la base source seront envoyées dans le fichier du cliché avant modification.

Cette technique, déjà employée par Oracle, permet de restreindre de manière drastique le verouillage pendant des opérations de lecture lourde. Si l'on imagine une base de données OLTP fortement sollicitée en exploitation courante et utilisée de temps à autre pour des extractions massives d'information telles que de l'analyse statistique de données (donc proche des techniques OLAP), alors, il y a tout intérêt à utiliser cette possibilité pour exécuter les requêtes d'analyse statistique sur un cliché de la base plutôt qu'en direct sur la base elle-même. Ceci minimisera les poses de verrous tout en permettant un plus grand parallélisme dans les traitements.

III-H - Base de données miroir

Il est possible de paramétrer une base de données pour qu'elle s'exécute en miroir, c'est à dire une copie absolue et synchrone des données sur un autre serveur. Dans ce cas un mécanisme de validation en deux phases fait que toute transaction opérée sur la base originale n'est assurément complète qu'après la validation sur la base miroir. Cela peut se faire avec ou sans serveur d'observation. Avec serveur d'observation le basculement est automatique et transparent. Le serveur d'observation peut être une version légère de MS SQL Server 2005 comme SQL Server Express (le remplaçant de MSDE).

III-I - Nouveaux paramètres de base de données

À la création, par défaut le pas d'incrément du grossissement des fichiers n'est plus de 10% mais de 1 Mo. En dehors de la possibilité de créer des bases de données snapshot (cliché) ou miroir, on trouve les paramètres suivants dans les ordres CREATE et/ou ALTER DATABASE :

EMERGENCY : base en mode lecture seule accessible sans login aux rôles fixe de serveur et aux membres du groupe sysadmin (maintenance en cas de base suspecte par exemple)

DB_CHAINING : permet le chaînage des privilèges entre différentes bases

TRUSTWORTHY : permet l'accès de modules dépersonnalisés (UDT par exemple) à des ressources externes à la base de données

AUTO_UPDATE_STATISTIC_ASYNC : permet qu'une requête n'attende pas la fin du calcul de mise à jour des statistiques pour s'exécuter.

PAGE_VERIFY (CHECKSUM) : mise en place d'une vérification de cohérence binaire des pages effectuée entre le disque et la mémoire par recalcul du code redondant (Attention : coûteux en temps et performances).

SUPPLEMENTAL_LOGGING : ajoute au journal des informations propres aux éditeurs d'add-on SQL Server 2005 (Attention : couteux en temps et performances)

DATE_CORRELATION_OPTIMIZATION : dans le cas d'une intégrité référentielle entre deux tables, basées sur au moins une colonne de type DATETIME, SQL Server maintient une corrélation statistique des données, permettant une plus grande optimisation des requêtes.

PARAMETRIZATION : permet de piloter le comportement de l'optimiseur de plans de requête lorsqu'il se trouve face à des requêtes comportant des valeurs scalaires pouvant être paramétrées.

III-J - Plan de requête

De nombreuses améliorations ont été apportées au plan de requête.

Désormais, le plan de requête peut être externalisé et réinjecté sous la forme d'un document XML. Cela permet de charcuter un tel plan (hauts risques) afin d'obliger un comportement particulier du moteur SQL. La visualisation du plan de requête sous forme XML se produit lorsque le flag STATISTICS XML est positionné à on :

```
SET STATISTICS XML ON
```

D'autre part, le moteur est maintenant capable de reconnaître des requêtes similaires contenant des valeurs scalaires différentes (paramètres) comme étant une même requête paramétrable. Dès lors si un plan de calcul a déjà été établi pour une valeur particulière des paramètres, il sera repris pour les autres valeurs de cette même requête.

Dans le même esprit un ordre SQL au sein d'une procédure peut maintenant faire l'objet d'une compilation et donc du calcul d'un plan de requête indépendant de l'ensemble de la procédure. Autrement dit, il s'agit d'un genre particulier de recompilation partielle.

III-K - Journal

Désormais le journal possède une architecture interne plus claire (virtual log files) permettant :

- d'éviter en cas de reprise après panne, de jouer les transactions finalement annulées (temps de reprise plus court);
- d'ajouter dans le journal même des informations en provenances d'add-on SQL server venant d'éditeurs tiers.

III-L - Envoi de mails

Ceux qui, comme moi, trouvaient complètement hallucinant de devoir ajouter un serveur Exchange pour envoyer un simple mail par SQL Server, seront désormais content de savoir que l'on peut envoyer un mail via la procédure stockée sendimail_sp, dont la syntaxe est assez poussée :


```
sendimail_sp [ [ @profile_name = ] 'profile_name' ]
[ , [ @recipients = ] 'recipients [ ; ...n]' ]
[ , [ @copy_recipients = ] 'copy_recipient [ ; ...n]' ]
[ , [ @blind_copy_recipients = ] 'blind_copy_recipient [ ; ...n]' ]
[ , [ @subject = ] 'subject' ]
[ , [ @body = ] 'body' ]
[ , [ @body_format = ] 'body_format' ]
[ , [ @importance = ] 'importance' ]
[ , [ @sensitivity = ] 'sensitivity' ]
[ , [ @file_attachments = ] 'attachment [ ; ...n]' ]
[ , [ @query = ] 'query' ]
[ , [ @execute_query_database = ] 'execute_query_database' ]
[ , [ @attach_query_result_as_file = ] attach_query_result_as_file ]
[ , [ @query_attachment_filename = ] query_attachment_filename ]
[ , [ @query_result_header = ] query_result_header ]
[ , [ @query_result_width = ] query_result_width ]
[ , [ @query_result_separator = ] 'query_result_separator' ]
```

```
[ , [ @exclude_query_output = ] exclude_query_output ]
```

Exemple

```
sendimail_sp
@profile_name = 'AdminMabase',
@recipients = 'sqlpro@microsoft.com',
@query = 'SELECT COUNT(*) FROM T_CHAMBRE
        WHERE CHB_COUCHAGE > 2 ,
@subject = 'Chambre de plus de 2 couchage',
@attach_query_result_as_file = 1 ;
```

Dans cet exemple, on envoie à sqlpro@microsoft.com avec le profil AdminMabase, le résultat de la requête dont la chaîne SQL est passée en paramètre. Certes il était possible de faire cela dans SQL Server 2000, à condition de savoir piloter des objets de type Ole à partir de commandes Transact SQL !

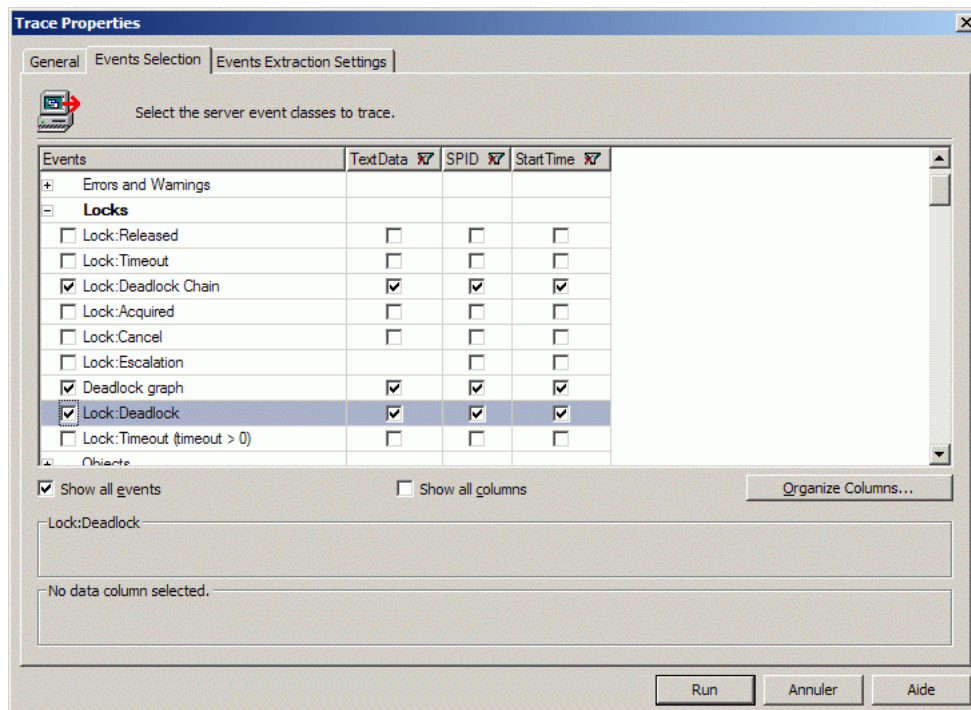
 *sendimail_sp n'est pris en charge qu'une fois que les objets de mailing aient été installés dans une base hôte de messagerie.*

III-M - Verrous mortels

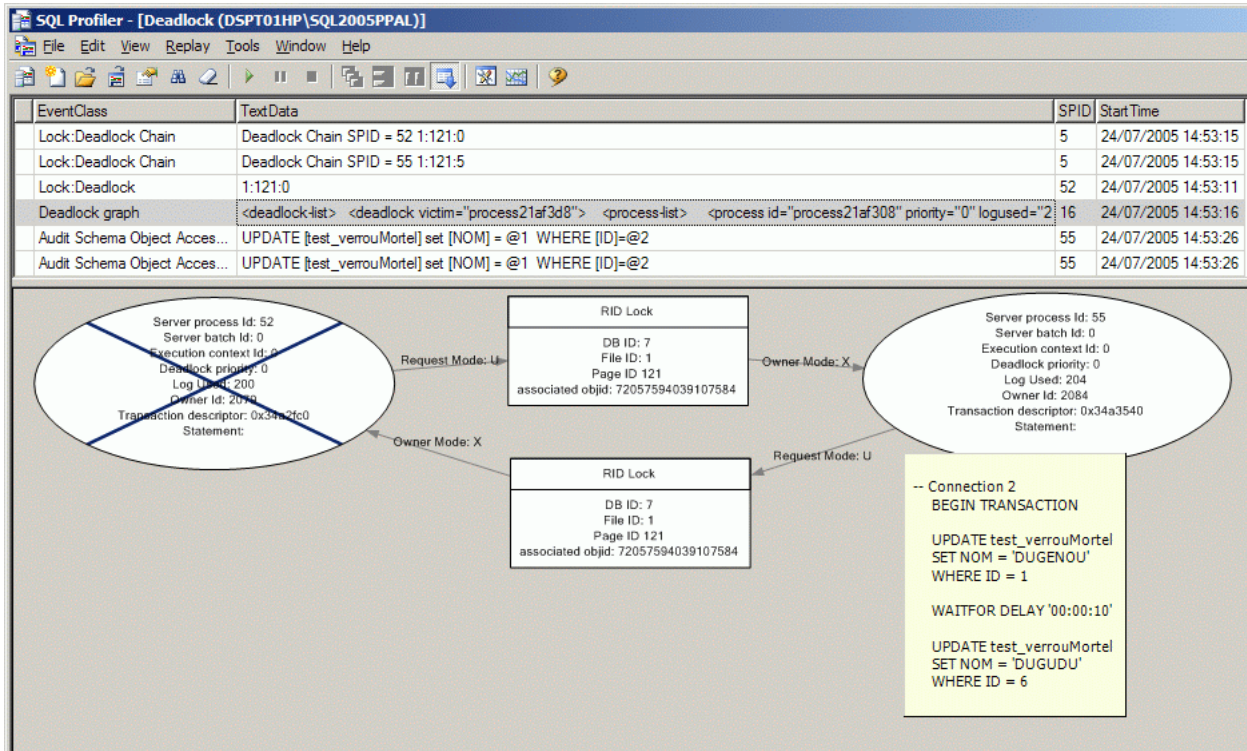
Un outil d'explication des verrous mortels (deadlocks, étreinte fatale, interblocage) a été mis en place. Il permet de visualiser le processus bloquant et les processus bloqués. Il est disponible dans le profiler SQL (trace). En activant dans le menu des événements à tracer, sous l'entrée "Locks" l'item "Deadlock Graph", vous disposez d'un affichage graphique des processus en jeu.

Voici un exemple de ce que cet outil restitue visuellement :

1) Activation du profiler avec suivi des événements de verrouillage mortel (deadlock, verrou mortel, étreinte fatale et interblocage sont des synonymes) :



2) Après avoir obtenu un verrou mortel, visualisez la trace dans le profiler :



3) Vous pouvez sauvegarder ces informations sous la forme d'un fichier xdl qui contient le graphique ci dessus au format XML :

```

<deadlock-list>
<deadlock victim="process21af3d8">
<process-list>
<process id="process21af308" priority="0" logused="204"
waitresource="RID: 7:1:121:5" waittime="12998" ownerId="2084"
transactionname="user_transaction"
lasttranstarted="2005-07-24T14:53:03.003" XDES="0x34a3540" lockMode="U"
schedulerid="1" kpid="2924" status="suspended" spid="55" sbid="0"
ecid="0" transcount="2" lastbatchstarted="2005-07-24T14:47:08.407"
lastbatchcompleted="2005-07-24T14:46:36.270"
clientapp="SQL Server Management Studio - Query" hostname="DSPT01HP"
hostpid="3280" loginname="sa" isolationlevel="read committed (2)"
xactid="236223201281" currentdb="7" lockTimeout="4294967295"
clientoption1="671090784" clientoption2="390200">
<executionStack>
<frame procname="adhoc" line="4" stmtstart="54"
sqlhandle="0x020000008a29270045c3cde9ab6112567469ffe4a4a34104">
UPDATE [test_verrouMortel] set [NOM] = @1 WHERE [ID]=@2 </frame>
<frame procname="adhoc" line="4" stmtstart="92" stmtend="240"
sqlhandle="0x02000000b5084a220d1ee556e23143e29580c1f2c3f2ca47">
UPDATE test_verrouMortel
SET NOM = &apos;DUGENOU&apos;
WHERE ID = 1 </frame>
</executionStack>
<inputbuf>
BEGIN TRANSACTION

UPDATE test_verrouMortel
SET NOM = &apos;DUGENOU&apos;
WHERE ID = 1

WAITFOR DELAY &apos;00:00:10&apos;

UPDATE test_verrouMortel
SET NOM = &apos;DUGUDU&apos;
WHERE ID = 6
</inputbuf>
</process>

```

```

<process id="process21af3d8" priority="0" logused="200"
  waitresource="RID: 7:1:121:0" waittime="4967" ownerId="2079"
  transactionname="user_transaction"
  lasttranstarted="2005-07-24T14:53:01.070" XDES="0x34a2fc0" lockMode="U"
  schedulerid="1" kpid="2852" status="suspended" spid="52" sbid="0"
  ecid="0" transcount="2" lastbatchstarted="2005-07-24T14:47:07.267"
  lastbatchcompleted="2005-07-24T14:46:50.290"
  clientapp="SQL Server Management Studio - Query" hostname="DSPT01HP"
  hostpid="3280" loginname="sa" isolationlevel="read committed (2)"
  xactid="223338299400" currentdb="7" lockTimeout="4294967295"
  clientoption1="671090784" clientoption2="390200">
  <executionStack>
    <frame procname="adhoc" line="13" stmtstart="54"
      sqlhandle="0x020000008a29270045c3cde9ab6112567469ffe4a4a34104">
UPDATE [test_verrouMortel] set [NOM] = @1 WHERE [ID]=@2      </frame>
    <frame procname="adhoc" line="13" stmtstart="330"
      sqlhandle="0x020000001454d004cc5495b1dbeb7411009ea2e13ef9460ce">
UPDATE test_verrouMortel
  SET NOM = &apos;DUMONT&apos;
  WHERE ID = 1      </frame>
  </executionStack>
  <inputbuf>
BEGIN TRANSACTION

UPDATE test_verrouMortel
SET NOM = &apos;DUHAMEL&apos;
WHERE ID = 6

WAITFOR DELAY &apos;00:00:10&apos;

UPDATE test_verrouMortel
SET NOM = &apos;DUMONT&apos;
WHERE ID = 1
  </inputbuf>
</process>
</process-list>
<resource-list>
<ridlock fileid="1" pageid="121" dbid="7"
  objectname="TEST.dbo.test_verrouMortel" id="lock2fb4c80" mode="X"
  associatedObjectId="72057594039107584">
  <owner-list>
    <owner id="process21af308" mode="X"/>
  </owner-list>
  <waiter-list>
    <waiter id="process21af3d8" mode="U" requestType="wait"/>
  </waiter-list>
</ridlock>
<ridlock fileid="1" pageid="121" dbid="7"
  objectname="TEST.dbo.test_verrouMortel" id="lock2fb4e40" mode="X"
  associatedObjectId="72057594039107584">
  <owner-list>
    <owner id="process21af3d8" mode="X"/>
  </owner-list>
  <waiter-list>
    <waiter id="process21af308" mode="U" requestType="wait"/>
  </waiter-list>
</ridlock>
</resource-list>
</deadlock>
</deadlock-list>

```

III-N - Synonymes

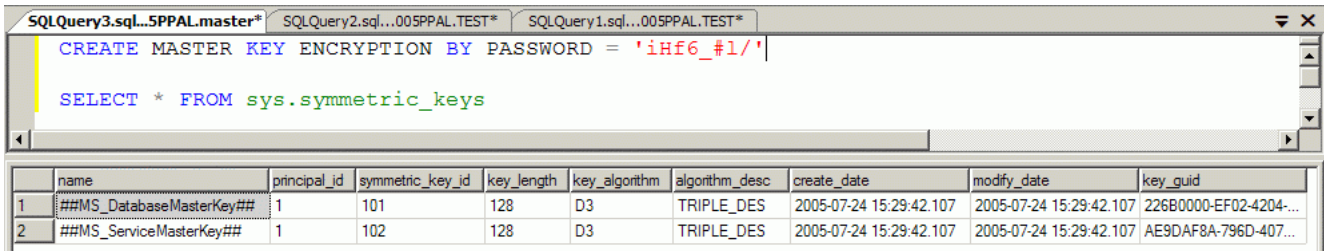
Afin de singer les pauvres et irrespectueuses limites d'Oracle en matière de noms d'objet SQL, MS SQL Server à introduit la mauvaise idée du concept de synonyme. Vous pouvez donc, même si c'est une horreur, donner un nom simple à n'importe quel objet de la base, c'est à dire aux procédure stockées, fonctions, tables et vues, et ainsi cintroduire la plus grande confusion dans votre base de données. La syntaxe pour ce faire est la suivante :

```
CREATE SYNONYM <nom_synonyme> FOR <nom_objet>
```

III-O - Cryptage

Le cryptage des données est désormais possible via l'utilisation de "certificats". Une clef maître de cryptage est créée lors de l'installation de MS SQL Server 2005. Elle a pour nom : "Service Master key". Chaque base de données doit avoir sa propre clef de chiffrement, que l'on obtient par la commande :

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '...'
```



The screenshot shows a SQL query window with the following commands:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ihf6_#1/'
SELECT * FROM sys.symmetric_keys
```

Below the query window, the results of the SELECT statement are displayed in a table:

	name	principal_id	symmetric_key_id	key_length	key_algorithm	algorithm_desc	create_date	modify_date	key_guid
1	##MS_DatabaseMasterKey##	1	101	128	D3	TRIPLE_DES	2005-07-24 15:29:42.107	2005-07-24 15:29:42.107	226B0000-EF02-4204-...
2	##MS_ServiceMasterKey##	1	102	128	D3	TRIPLE_DES	2005-07-24 15:29:42.107	2005-07-24 15:29:42.107	AE9DAF8A-796D-407...

La vue système sys.symmetric_keys, contient les informations des clefs de chiffrement.

Une fois cette clef créée, vous pouvez obtenir un certificat (ou une clef asymétrique) qui vous permettra de crypter les données que vous avez à véhiculer. Le certificat s'obtient par la commande :

```
CREATE CERTIFICAT <nom_certif>
WITH SUBJECT = '<libelle>'
EXPIRY_DATE = '<date_expiration>'
```

Notez qu'un certificat n'est donc valable qu'un certain temps...

Pour crypter et décrypter des données, vous pouvez utiliser les fonctions suivantes :

EncryptByCert()	DecryptByCert()
EncryptByAsymKey()	DecryptByAsymKey()
EncryptByKey()	DecryptByKey()
EncryptByPassphrase()	DecryptByPassphrase()

En sus, la fonction Cert_id('<nom_certif>') renvoie l'identifiant du certificat en passant son nom comme paramètre.

Exemple

```
CREATE CERTIFICAT SQLpro
WITH SUBJECT = 'Fred Brouard'
EXPIRY_DATE = '20050831'

EncryptByCert(Cert_id('SQLpro'), 'Ma phrase à crypter...')
```

Le cryptage des données est nécessaire notamment pour l'utilisations des Web services via SQL Service Broker notamment dans les phases d'authentification et les processus de messagerie. En revanche le cryptage des données à l'intérieur d'une base ne relève que peu d'intérêt.


A lire l'excellent article de **Marcin Policht sur le sujet**.

III-P - Dépersonnalisation (Execute AS)

SQL Server 2005 permet de définir implicitement le contexte d'exécution d'une procédure, une fonction (sauf table en ligne), un trigger (y compris DDL) ou une file d'attente (QUEUE dans Service Broker). Cette technique requiert l'utilisation du mot clef EXECUTE AS avec en paramètre le nom d'utilisateur (ou de connexion dans la cas d'un trigger DDL de serveur, ou bien à l'aide des paramètres objet CALLER, SELF ou OWNER. Dès lors la sécurité d'exécution et son contexte sont codé dans l'objet. Cela permet par exemple de donner des privilèges et donc de gérer une sécurité fine à des commande qui sont dépourvues de sécurité, comme la commande TRUNCATE.

Exemple

```
CREATE PROCEDURE P_TRUNCATE (@A_TABLE NVARCHAR(128))
WITH EXECUTE AS 'SQL_troncator'
AS
    DECLARE @QUERY NVARCHAR(150)
    SET @QUERY = 'TRUNCATE TABLE ' + @A_TABLE
    EXECUTE (@SQL)
```

 La fonction *REVERT* permet de revenir au contexte d'exécution de l'appelant.

III-Q - Conclusion

Les tables réparties comme les manipulations d'index en ligne et le snapshot, comme le miroir de bases sont des nouveautés visant les VLDB (Very Large Data Bases), marché dans lequel SQL Server s'est finalement bien introduit comme en témoigne **WinterCorp**. La gestion des exceptions, autrefois point noir du codage Transact SQL hérite des techniques des langages modernes et c'est un plus indéniable. Les triggers de DLL apportent de bonnes solutions à des problématiques complexes, notamment à l'absence d'assertion (norme SQL), encore faudrait-il n'en point abuser. La procédure *sendmail_sp* est un bienfait à l'horrible verrue que constituait l'envoi de mail par les versions antérieures de SQL Server. Quant à plan de requête en XML, l'avenir dira comment il sera utilisé !

Bref, presque rien que du bon, à l'exception de la stupide introduction des synonymes pour singer Oracle.

IV - Conclusion sur MS SQL Server 2005

Comme l'exprime très bien Joe Celko, dans son article "**Keys to the Database**", je ne suis pas super enthousiaste à l'arrivée de SQLCLR, c'est à dire de procédures stockées écrites en .net, au sein de la base de données et exécutées sur le serveur...

Non pas que je trouve le langage C# mauvais, bien au contraire. Celui qui l'a écrit fait partie de mes "fans" et le style de la chose est plutôt parmi les projets intelligents comme le furent le langage ObjectPal de Paradox sous Windows (trop en avance dans ses concepts hélas), ou Delphi et son succès mérité (tiens, trois produits du même auteur ???). Non, là où le bât blesse, c'est que, parcourant le net à la recherche de code SQLCLR pour expliquer ces concepts, je tombe deux fois sur trois sur du VB .net horriblement mal écrit. Et comment pourrait-il en être autrement avec un langage "à tout faire" écrit à l'origine pour les "débutants" (BASIC : **Beginner**'s All-purpose Symbolic Instruction Code)...

Que dire aussi du défaut d'impédance (NULL, collations, typage...) introduit par les L4G comme C# ? Je m'étonne par exemple à chaque fois de la complexité des solutions proposées pour résoudre ce problème dans les langages autres que SQL, et je m'étonne encore plus de l'absence quasi systématique de cette question dans le code que j'audite...

Bref, et comme me le disait un professeur d'université, "*l'avantage c'est que n'importe quel couillon va être capable de pisser du C# pour faire sa petite fonction. L'inconvénient, c'est quand la fonction va devoir être appelée quelque centaines de milliers de fois dans les lignes d'une table balayée par un SELECT !*". Car il y a fort à parier, comme c'est le cas actuellement dans les projets employant SQL Server, que peu d'entreprise mettrons un DBA costaud pour valider le code à intégrer dans la base...

D'autant que SQLCLR ne remplace pas Transact-SQL. En avril 2004, invité à titre de MVP chez Microsoft à Redmond pour discuter de la chose, nombre de MVP avait montré leur mécontentement : C# s'avérait moins bon que Transact SQL dans 2 cas sur 3 (c'est toujours le cas), mais l'aspect mercatique étant le plus fort, les microsoftees ne voulait démordre de cette stratégie. Le tohu bohu fut tel que la plupart des présentations que les équipes de développement de MS voulait nous montrer, s'arrêtaient à la deuxième diapo et il s'ensuivait des débats soutenus mais souvent constructifs. La conclusion fût amusante : certains MVP facétieux avait réalisé un tee-shirt sur lequel on voyait un "SQL Server" barré, remplacé par un rageur "CLR Server" !

La contrepartie de la systématisation de .net au sein de SQL Server c'est que l'on l'on va devant un marché sans doute juteux : MS sql Server 2005 va faire un tabac, et des gens comme moi vont devoir intervenir à chaud, lorsqu'au

bout de quelques mois d'exploitation, la volumétrie tirera les performances vers le bas et que l'on saura pas comment optimiser l'application, c'est à dire rectifier les horreurs qui ont été écrites par des développeurs à la culture "base de données" restreinte !

V - Références

Livres :

A first look at SQL Server 2005 for Developers - Bob Beauchemin, Niels Berglund, Dans Sullivan - Addison Wesley 2004

Microsoft SQL Server 2005 New Features - Michael Otey - McGraw-Hill Osborne 2004

Web :

<http://www.microsoft.com/France/sql/sql2005/decouvrez/fonctionnalites.msp>

<http://morpheus.developpez.com/SQL-Server-2005/>

<http://www.sqlservercentral.com/>

<http://www.extremeexperts.com/>

<http://www.sqljunkies.com/>