

# Maintenance des index dans les VLDBs (*Very Large Databases*)



par Frédéric Brouard, alias SQLpro  
*MVP SQL Server*  
*Expert langage SQL, SGBDR, modélisation de données*

Auteur de :

- SQLpro <http://sqlpro.developpez.com/>
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseignant aux Arts & Métiers et à l'ISEN Toulon

*L'entretien des index est une condition essentielle et primordiale de maintien des performances des requêtes. Voici une petite étude qui fait le point sur ce qu'est un index, comment il vieillit et comment on procède à sa maintenance dans le cadre des VLDB dont le service est en principe continu...*

Copyright et droits d'auteurs : La Loi du 11 mars 1957 n'autorisant aux termes des alinéas 2 et 3 de l'article 41, d'une part que *des copies ou reproductions strictement réservées à l'usage privé et non [...] à une utilisation collective*, et d'autre part que les analyses et courtes citations dans un but d'illustration, toute reproduction intégrale ou partielle faite sans le consentement de l'auteur [...] est illicite. Le présent article étant la propriété intellectuelle de Frédéric Brouard, prière de contacter l'auteur pour toute demande d'utilisation, autre que prévu par la Loi à [SQLpro@SQLspot.com](mailto:SQLpro@SQLspot.com)

## 0 - Qu'est ce qu'un index ?

Par sa nature un index est une information redondante qui n'est en aucun cas nécessaire à la logique relationnelle (c'est d'ailleurs pourquoi les index n'existent pas dans le langage SQL).

Par analogie, il suffit de dire la ville ou vous habitez pour vous retrouver. Sans précision de rue la recherche sera longue (il suffit de visiter toutes les maisons de la ville). Avec le nom de rue (dont les nomenclatures ont commencés au moyen âge) cette recherche sera déjà moins longue. Avec le numéro dans la rue (notion datant de la révolution) ce sera quasi immédiat. Voici donc un index que vous utilisez tous les jours sans le savoir. Et pour la poste, le code postal est un index encore plus puissant puisqu'il permet d'effectuer un tri automatique du courrier...

Même chose pour le téléphone. Au début de son invention, les demoiselles (du téléphone) connectais les quelques abonnés par référence à leur nom dans une ville. Puis vint une première numérotation par ville. C'est ainsi que Fernand Raynaud avait du mal à obtenir le [22 à Asnières](#) ! Voir <http://www.youtube.com/watch?v=QHWqaSdVbnY>  
Finalement les transformations du téléphone à partir des années 70 et jusqu'à nos jours permettent aujourd'hui d'accéder à quiconque sur la planète à partir d'un simple numéro, reléguant ainsi les opératrices au rangs d'assistantes pour la recherche dans les annuaires !

On constate donc qu'un index n'est pas nécessaire à l'information, mais permet une efficacité de recherche optimale (ce pourquoi il est créé d'ailleurs).  
Seul inconvénient, il constitue une sorte de redondance dans le sens ou il oblige à stocker plus de données que l'information basique...

Un index SQL Server est créé derrière chaque contrainte de type PRIMARY KEY ou UNIQUE. Il n'y a pas d'index derrière une FOREIGN KEY.  
Les limites des index SQL Server sont les suivantes : au maximum 16 colonnes avec une longueur de clef de 900 octets au plus. En outre, il ne peut y avoir plus de 250 index par table. Enfin SQL Server accepte de créer des index sur des colonnes calculées (le calcul doit être déterministe) et sur des vues (dans ce dernier cas il y a des limitations importantes).

## 1 - Structure des index

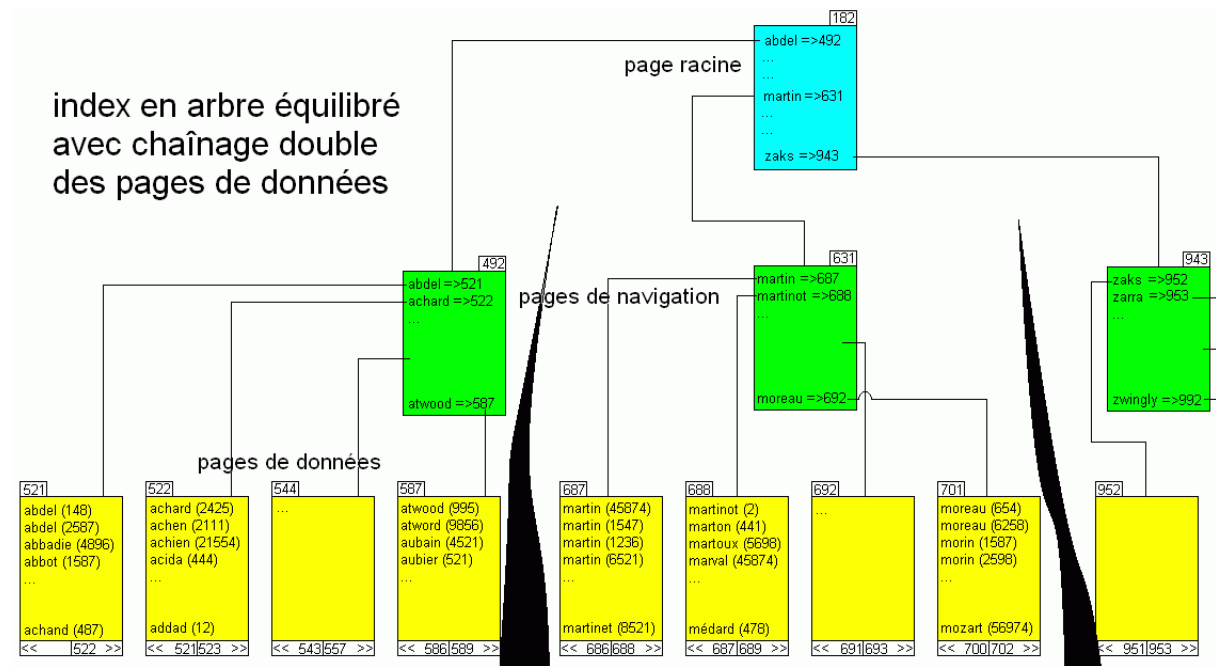
Rappelons qu'un index est constitué dans MS SQL Server par un assemblages de pages de 8 Ko, chaque page contenant un nombre de ligne de la table ou de l'index.  
Rappelons aussi qu'une clef d'index, c'est à dire la donnée indexée et les données techniques nécessaire à la clef représente une ligne d'index.

Tous les index de MS SQL Server sont des index de type B-Tree, c'est à dire des arbres équilibrés.

### 1.1 - index B-Tree

Un arbre équilibré est une structure de données arborescente dont les feuilles sont toutes situées au même niveau. Le nombre de niveau d'un B-Tree (Balanced Tree) est la caractéristique principale de ce type d'arbre.

Chaque page d'aiguillage d'un B-Tree permet de se brancher sur un nombre de page variable, les données d'aiguillage à l'intérieur de la page étant triées.  
Voici un exemple de ce qu'est un arbre équilibré d'index :



Notez dans cet index que la page racine contient de nombreuses "lignes" d'index (appelées en fait clef d'index) contenant à la fois une valeur d'aiguillage et l'adresse de la page où trouver la valeur recherchée.

Les pages racine (en bleu) et navigation (en vert) ne contiennent que des valeurs d'aiguillage. Seule les pages feuilles (en jaune) contiennent les données réelles.

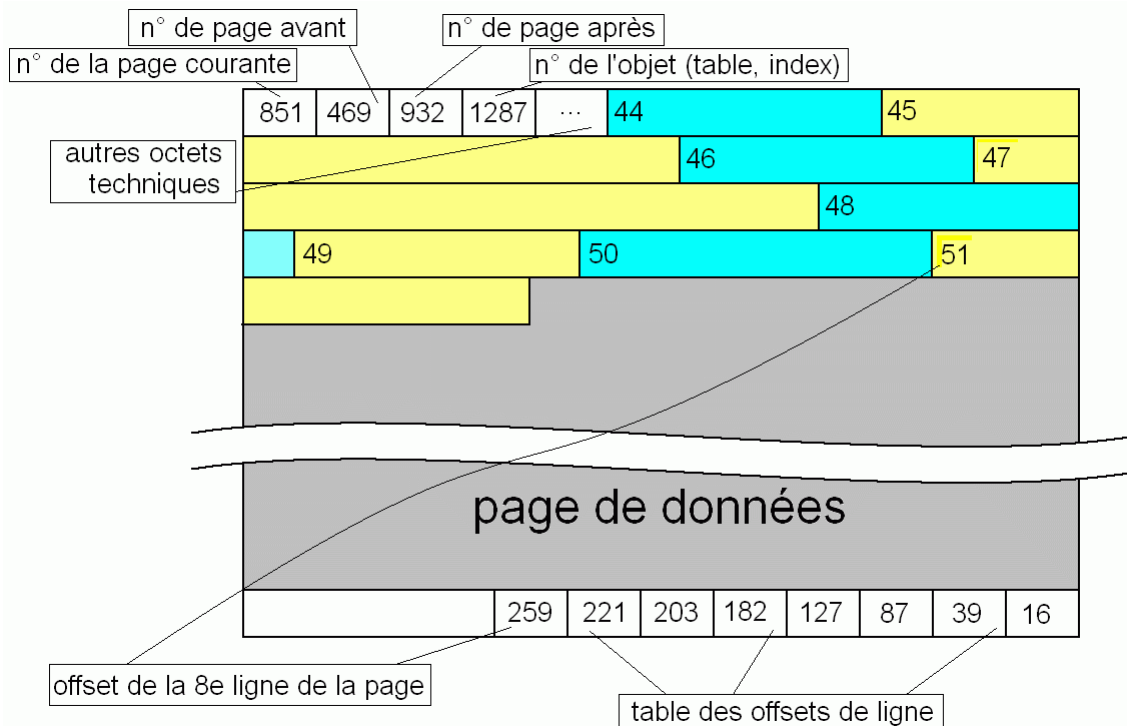
Il peut y avoir autant de niveau que nécessaire à l'index.

## 1.2 - Pages d'index

Les pages, toutes de 8Ko, sont structurées comme suit...

Chaque page possède :

- quelques octets techniques (96) comportant notamment :
  - un numéro l'identifiant.
  - la référence à l'objet contenu (table ou index),
  - le nombre d'octets libres,
  - les références aux pages précédente et suivante...
- une zone avec le contenu des lignes (appelés "slots")
- une table des offset de slots de ligne en ordre inverse.



Cette manière d'organiser la page avec les lignes qui s'incrivent en descente dans la page et la table des offsets de ligne qui s'allonge en montant dans la page permet d'en optimiser le remplissage. Cette technique est rendue nécessaire parce que nous ne savons pas à l'avance quel est le nombre de ligne d'une table ni même la longueur de chaque ligne du fait des types de données de taille variable de SQL (varchar par exemple...).

Les offset de ligne sont de 2 octets.

Une ligne commence par une entête de 16 octets technique comportant entre autres : le type de ligne, le nombre de colonnes, debut de la zone fixe / variable, une matrice de nullabilité.... C'est pour cette raison que la taille d'une ligne ne peut dépasser 8060 octets.

Pour plus de détails, voir :

- <http://www.sqlskills.com/blogs/paul/2007/09/30/InsideTheStorageEngineAnatomyOfARecord.aspx>
- <http://www.sqlskills.com/blogs/paul/2007/10/03/InsideTheStorageEngineAnatomyOfAPage.aspx>
- <http://209.34.241.67/sqlserverstorageengine/default.aspx?p=7>

### 1.3 - clef d'index

Une clef d'index est constitué par une valeur et un numéro de page. Elle sert à indiquer où se trouve la page suivante contenant soit la donnée, soit de nouvelles clefs d'index si l'on ne se trouve pas encore sur les pages feuilles.

Dans la page racine comme dans les pages de navigation, il n'y a que des clefs d'index. Dans les pages feuilles on ne trouve que des pages de données.

Une clef d'index est donc constitué d'une valeur de clef et d'une référence de page (4 octets). La longueur de la clef, ne doit pas en principe dépasser 900 octets.

## 1.4 - données d'index

Les données d'index sont situées dans les pages feuilles c'est à dire au dernier niveau (en bas) de l'index. Ces données font elle-même référence à la ligne de la table contenant les données complémentaires.

### Exemple :

Soit un index posé sur un nom de personne dont la longueur moyenne est de 8 caractères.  
Soit une table comportant 2 milliards de lignes (c'est déjà beaucoup !).  
Combien de pages doivent nécessairement être lues pour trouver un nom comme "dupont" ?

Calcul :

la clef d'index est constituée de 8 caractères, soit 8 octets + 4 octets pour indiquer la page suivante + 2 octets pour l'offset de ligne + 16 octets techniques de ligne. Total : 26 octets.  
L'espace réservé aux lignes dans la page étant de 8060 octet, il y a donc  $8060 / 26 = 310$  lignes dans chaque page.

Voici à chaque niveau le nombre de page indexées et le nombre total de pages :

Niveau	Références	nb page du niveau	nb page total	nb lignes au niveau
1 (racine)	1 pages poite vers 310 pages	1		310
2 (navigation)	310 pages pointent vers 310 pages chacune	310	311	96100
3 (navigation)	96100 pages pointent vers 310 pages chacune	96 100	96411	29 791 000
4 (navigation)	29 791 000 pages pointent vers 310 pages chacune	29 791 000	29 887 411	9 235 210 000

Il n'est pas nécessaire de "descendre" plus bas car au 4e niveau nous avons déjà plus de 9 milliards de lignes référencées.

Le quatrième niveau sera donc constitué de 2 milliards de ligne / 310 = 6 451 613 de pages contenant en fait les données et non plus des valeurs d'aiguillage (*en fait nous faisons une petite approximation qui sera rectifié lorsque nous aurons vu le paragraphe suivant*).

L'index sera donc constitué comme suit :

Niveau	Références	nb page du niveau	nb page total	nb lignes au niveau
1 (racine)	1 page pointe vers 310 pages	1		310
2 (navigation)	310 pages pointent vers 310 pages chacune	310	311	96100
3 (navigation)	96 100 pages pointent vers 310 pages chacune	96 100	96411	29 791 000
4 (navigation)	6 451 613 pages contenant les données	6 451 613	6 548 024	2 000 000 000

Rechercher un nom comme "Dupont" nécessite donc de lire 4 pages depuis la racine jusqu'a la page de données contenant ce nom.

En imaginant que la table non indexée soit constituée des colonnes ID, nom, prenom, date naissance, c'est à dire environ 30 octets en moyenne – *plus les octets techniques, soit 52 octets* – il faut environ 12 903 226 pages pour stocker la table.

Le rapport entre lecture de la table par balayage de toutes les pages et recherche par aiguillage dans l'index est donc de  $12\,903\,226 / 4 = 3\,225\,806,5\dots$

Autrement dit le nombre de page à lire par balayage de la table est 3 millions de fois plus couteux que la recherche par l'index...

On comprendra donc l'importance et la nécessité d'une bonne indexation, surtout dans les grandes tables !

## 1.5 - Index cluster et heap

Il existe 2 formes d'index dans SQL Server les index cluster et les index en heap.

L'index **cluster** est en fait un index greffé sur la table elle même. Au niveau des pages de navigation on en trouve que la clef d'index. Au niveau des pages de données on trouve la totalité des colonnes de la table. Le mot "cluster" signifie emplacement ce qui est approprié car il indique que les lignes de la table sont physiquement triée.

Comme l'index cluster est en fait la table elle même, il ne peut y avoir qu'un seul index cluster par table.

Qu'une table possède un index cluster est un avantage, car cela économise de la place puisqu'une redondance est évité. C'est pourquoi lors de la création de l'index primaire SQL Server utilise systématiquement un index cluster. On peut obliger SQL Server à utiliser un index heap pour la PRIMARY KEY en spécifiant :

```
PRIMARY KEY NONCLUSTERED
```

L'index en **heap** est en fait un index indépendant de la table. SQL Server permet d'en poser 249 par table.

Afin de pouvoir retrouver de quelle ligne vient l'information indexée, dans les pages de type feuille, c'est à dire celles contenant des données, une information supplémentaire est ajoutée :

- soit la valeur de la clef primaire si la table en possède une
- soit un trio de données composé de l'identifiant de fichier (entier 2 octets) + l'identifiant de page (4 octets) + l'identifiant de slot de ligne (2octets), si la table ne possède pas de clef primaire

Pour une contrainte d'unicité, SQL Server pose systématiquement un index heap. On peut obliger SQL Server à poser un index cluster sur une contrainte d'unicité ou lors de la création d'un index quelconque en précisant le mot clef CLUSTER.

Pour toutes ses raisons, il conviendrait que toute table ait un index cluster. Mais le choix de l'index cluster est primordial. Mieux vaut une clef d'index de petite taille et dont la valeur ne change jamais. Un auto incrément est tout indiqué !

## 1.6 - Lecture des index

SQL Server ne lit pas les données par page, mais par bloc de 8 pages appelées *extensions* (extents en anglais). En effet ne lire que 8 Ko de données est relativement faible compte tenu de l'inertie de la tête de lecture du disque. Il en est de même pour l'allocation d'espace dans les fichiers qui se fait par un multiple de l'extension, soit n fois 64 Ko.

## 1.7 - index et tables

Notons que SQL Server utilise le terme d'index aussi bien pour des structures de tables que pour les index. Tant est si bien que nous avons quatre types de structures de stockage de données possible :

- les index cluster (en fait des tables)
- les lignes de tables, pour des tables n'ayant pas d'index cluster
- les index heap
- les blobs (c'est à dire les longs objets binaires qui sont stockés hors de pages de lignes de tables)

## 2 - Pourquoi reindexer ?

Au fur et à mesure que la base de données "vit" les structures de stockage des lignes de la table, comme celles des index se détériorent. En effet chaque mise à jour, insertion ou suppression provoque des réorganisations de pages ou de ligne qui rendent les index moins efficaces. Si l'on agit pas pour rectifier cette désorganisation, l'efficacité des index en pâtit.

### 2.1 - Split de page

Un "split" de page, c'est à dire la séparation d'une page de données contenant des lignes de table ou d'index en deux pages, se produit lorsque l'on doit insérer une nouvelle ligne dans une page pleine.

Dans ce cas, la page cible est coupée en deux en son milieu. Les lignes du haut sont conservées dans la page actuelle et les lignes du bas sont déplacées dans la nouvelle page créée. Ainsi au lieu d'une page on se trouve avec deux pages comportant un nombre égal de ligne et la nouvelle ligne est insérée au bon endroit, c'est à dire dans l'ordre prévu par l'index. De ce fait chacune des deux nouvelles pages est rempli à la moitié.

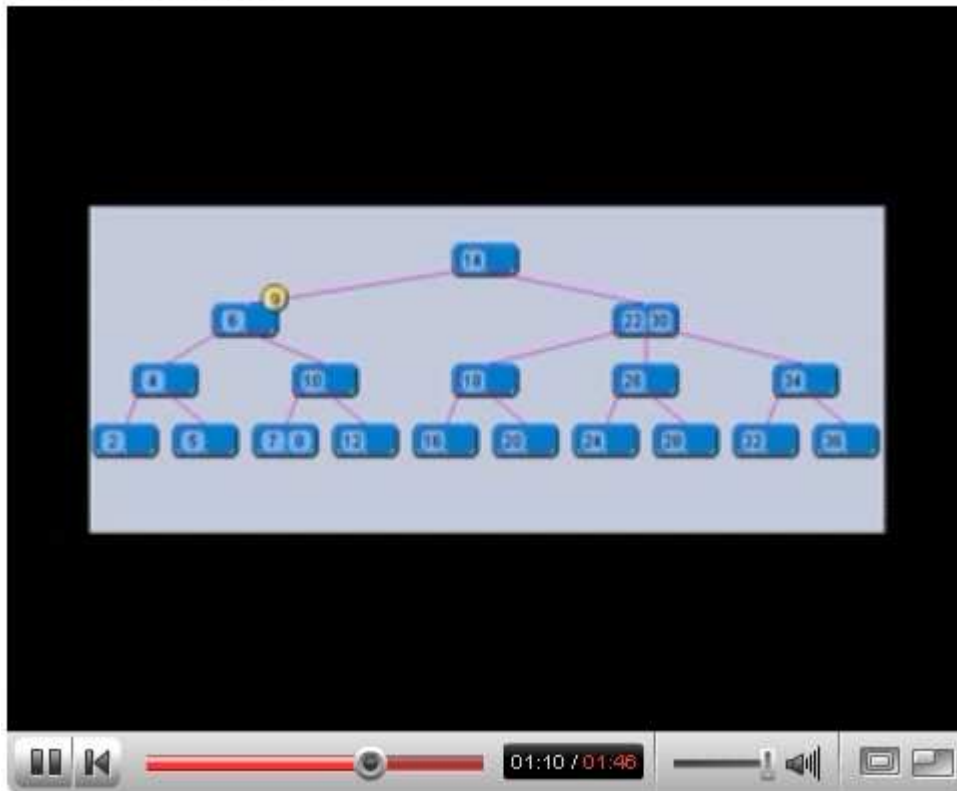
En répétant ce phénomène souvent, l'index grossit et se fragmente fortement. En effet à chaque split de page génère une page supplémentaire, chacun des deux étant remplis à moitié...

Le phénomène observé au niveau des feuilles de l'arbre peut d'ailleurs se reproduire dans les pages supérieurs (pages mères de référence...), jusqu'à obliger à l'ajout d'un nouveau niveau à l'arbre.

L'algorithme en jeu est complexe et couteux. Mieux vaut que le nombre de split de page soit assez bas. Nous verrons comment faire pour éviter ce phénomène.

Un bon exemple de ce qui se passe dans un index est présent dans la video :

## B-Tree example



<http://www.youtube.com/watch?v=coRJrcIYbF4>

On peut se rendre compte des effets dévastateurs du split de page à l'aide du script SQL suivant :

```
CREATE DATABASE DB_INDEXES;
GO

USE DB_INDEXES;
GO

CREATE TABLE T1 (K INT IDENTITY, C CHAR(800));
GO

CREATE INDEX X ON T1 (C);
GO

-- boucle d'insertion de données aléatoires
DECLARE @I INT
SET @I = 0
WHILE @I < 10000
BEGIN
    INSERT INTO T1 VALUES (REPLICATE(CHAR(CAST(FLOOR((RAND() * 93) + 32) AS INT)),
    800))
    SET @I = @I + 1
END

-- test
SELECT TOP 100 * FROM T1

-- mesure de la fragmentation :
DBCC SHOWCONTIG(T1, X) WITH TABLERESULTS, ALL_LEVELS
```

Voici un extrait du résultat de la commande ci dessus :

Level	Pages	Rows	AverageFreeBytes	AveragePageDensity
0	1661	10000	3195.3369140625	60.522151947021484
1	287	1661	3350.284912109375	58.607795715332031
2	49	287	3293.14208984375	59.313785552978516
3	9	49	3631.554931640625	55.132755279541016
4	2	9	4406.0	45.564617156982422
5	1	2	6456.0	20.237213134765625

Les données de fragmentation de l'index X montre que les pages sont de plus en plus vides au fur et à mesure que l'on descend dans les niveaux de l'index. Ceci est dû à l'espace libre laissé par les multiples split de page. L'index est peu dense et coûte cher en page donc à la lecture.

Cet index consomme actuellement 2009 pages. Après remaniement (reconstruction de l'index pour éliminer la fragmentation) il ne mesure plus que 1112 pages ! En gros, une fois nettoyé cet index fonctionne deux fois mieux...

Notons que ce phénomène se produit à l'insertion (INSERT).

## 1.2 - Déplacement de lignes

Le déplacement des lignes d'une table (lorsqu'elle est organisée en cluster) ou d'un index est lié à l'utilisation des type de données SQL de taille variable.

Comme on l'a vu, dans une page d'un fichier de données de la base, les lignes sont enchâssées, c'est à dire qu'à une ligne succède une autre sans aucune perte de place.

Que se passe t-il si une mise à jour de ligne agrandit la ligne ?

Prenons par exemple un client dont l'adresse codée exclusivement en VARCHAR change pour passer de :

13 rue Liberté 83670 BARJOLS

à

144 avenue du Maréchal de Lattre de Tassigny 83945 SEILLON SOURCE D'ARGENS

?

Une page de la table T\_CLIENT :

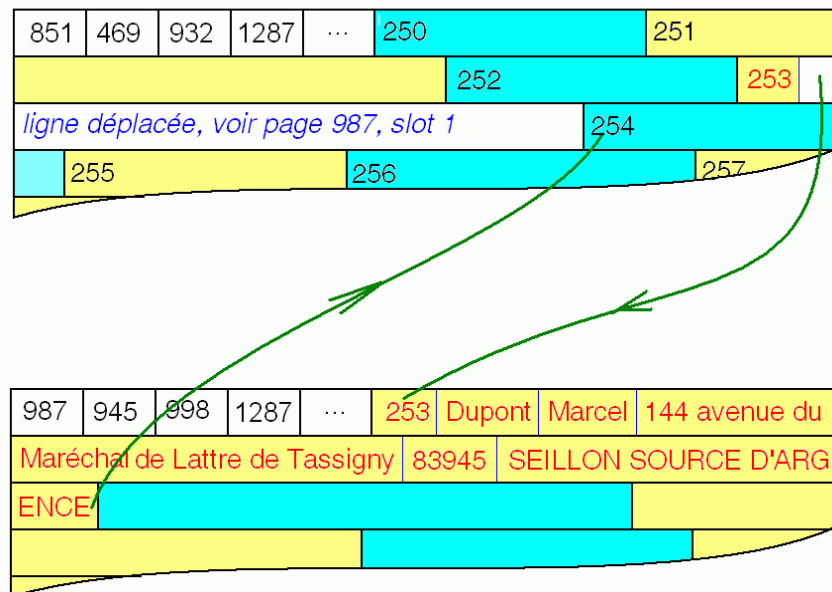
851	469	932	1287	...	250	251
					252	253 Du
pont Marcel		13 rue Liberté	83670	BARJOLS	254	
255		256			257	

```
UPDATE T_CLIENT_CLI
SET CLI_ADRESSE_RUE = '144 avenue du Maréchal de Lattre de Tassigny',
    CLI_CODEPOSTAL = 83945,
    CLI_VILLE = 'SEILLON SOURCE D'ARGENCE'
WHERE CLI_NUM = 253
```

La ligne ne pouvant plus être stockée à l'endroit initial par faute de place, elle doit être mise ailleurs. Néanmoins, comme cette ligne provient d'un index (par exemple l'index cluster

constituant la table), on ne peut pas bousculer l'ordre physique des lignes. Un bon compromis est de laisser la clef de l'index à l'emplacement initial et de faire référence à la nouvelle ligne à la place des données complémentaires.

Déplacement d'une référence de ligne suite à un UPDATE élargissant la ligne



La nouvelle ligne aura un pointeur indiquant le retour à l'emplacement initial.

### Démonstration

```
CREATE DATABASE DB_INDEXES;
GO

USE DB_INDEXES;
GO

CREATE TABLE T1 (K INT NOT NULL IDENTITY PRIMARY KEY, C VARCHAR(800));
GO

-- boucle d'insertion de données aléatoires
DECLARE @I INT
SET @I = 0
WHILE @I < 10000
BEGIN
    INSERT INTO T1 VALUES (REPLICATE(CHAR(CAST(FLOOR((RAND() * 93) + 32) AS INT)),
    100))
    SET @I = @I + 1
END

-- mise à jour de la table sur 10% des lignes avec des données agrandies
UPDATE T1
SET C = C + REPLICATE('*', 700)
WHERE K % 10 = 0

-- test
SELECT TOP 100 * FROM T1
```

```
DBCC SHOWCONTIG(T1, 0) WITH TABLERESULTS
```

Le résultat montre que le nombre de passage d'une extension à l'autre (extentSwitches) est démesuré comparé au nombre des extensions. En effet, selon la règle des piquets et des intervalles, pour passer d'un piquet à l'autre sur  $n$  piquets il faut  $n-1$  intervalle... Nous sommes très loin du compte :

Pages	Rows	Extents	ExtentSwitches	BestCount	ActualCount	ExtentFrag
290	10000	38	288	37	289	2.63157892

Cette disproportion est due au fait des lignes déplacées qui oblige à lire une page s'interrompre au milieu, aller lire une autre page puis revenir... et ainsi de suite pour toutes les lignes dont la taille a évolué.

Ce phénomène se produit lors des modifications de ligne (UPDATE) pour des données de taille variables. Il engendre non seulement des espaces vides, mais aussi oblige des lectures en zigzag...

Une fois nettoyé, l'index ne requiert plus que 250 pages sur 32 extensions et donc commute 31 fois pour un balayage de la table au lieu de 288. Bref la maintenance de cet index a permis d'améliorer plus de 9 fois...

### 1.3 - Ligne fantôme

Dès lors qu'il y a suppression, la ligne libérée, qu'elle soit d'index ou de table ne l'est que logiquement. Aucune place n'est libérée sauf si cette place peut être récupérée pour une nouvelle ligne, qui, par un grand hasard, pourrait avoir les mêmes valeurs de clef que celle supprimée... Les lignes ainsi supprimées logiquement sont appelées *lignes fantômes* ("ghost record" en anglais).

Voici un script SQL permettant de mettre en évidence ce problème :

```
CREATE DATABASE DB_INDEXES;
GO

USE DB_INDEXES;
GO

CREATE TABLE T1 (K INT NOT NULL IDENTITY CONSTRAINT PK PRIMARY KEY, C
VARCHAR(800));
GO

-- boucle d'insertion de données aléatoires
DECLARE @I INT
SET @I = 0
WHILE @I < 10000
BEGIN
    INSERT INTO T1 VALUES (REPLICATE(CHAR(CAST(FLOOR((RAND() * 93) + 32) AS INT)),
800))
    SET @I = @I + 1
END

-- suppression de 10% des lignes
DELETE FROM T1
WHERE K % 10 = 0
```

```
-- vérification des données dans les pages :
DBCC TRACEON (3604);
GO

DECLARE @first BINARY(6), @fichier INT, @page INT, @SQL NVARCHAR(4000);
SELECT @first = first
FROM sysindexes
WHERE name = 'PK'
SELECT @page = CAST( SUBSTRING(@first, 4, 1) + SUBSTRING(@first, 3, 1)
                    + SUBSTRING(@first, 2, 1) + SUBSTRING(@first, 1, 1) AS INT),
       @fichier = CAST(SUBSTRING(@first, 6, 1) + SUBSTRING(@first, 5, 1) AS INT);

SET @SQL = 'DBCC PAGE(DB_INDEXES, '
          + CAST(@fichier AS NVARCHAR(16)) + ', '
          + CAST(@page + 1 AS NVARCHAR(16)) + ', 1)'
EXEC (@SQL)
```

Dans l'onglet Message, vous allez voir apparaître divers données concernant la page. En particulier le slot 0 de cette page présente une ligne fantôme (Record Type = GHOST\_DATA\_RECORD) :

```
PAGE: (1:29)
-----

BUFFER:
-----

BUF @0x00F9B200
-----
bpage = 0x47900000      bhash = 0x00000000      bpageno = (1:29)
...

PAGE HEADER:
-----

Page @0x47900000
-----
m_pageId = (1:29)      m_headerVersion = 1      m_type = 1
...

Allocation Status
-----
GAM (1:2) = ALLOCATED      SGAM (1:3) = NOT ALLOCATED
...

DATA:
-----

Slot 0, Offset 0x60
-----
Record Type = GHOST_DATA_RECORD
Record Attributes = NULL_BITMAP VARIABLE_COLUMNS
47900060: 0008003c 0000000a 01000002 6b032f00 <...../..k
47900070: 6b6b6b6b 6b6b6b6b 6b6b6b6b 6b6b6b6b kkkkkkkkkkkkkkkk
...
```

Bien entendu on peut mesurer l'espace logique libéré avec la commande DBCC. Cela se voit dans le nombre d'octets libre des pages.

### 3 - Comment voir la fragmentation ?

Comme nous venons de le voir, les mises à jour pénalisent les index. Cela se traduit par de la fragmentation à différents niveaux...

Les insertions et insertions provoquent de la fragmentation de page, les modifications provoquent de la fragmentation d'extension.

Tout ceci peut être mesuré à l'aide de différents moyens.

#### 3.1 - Voir la fragmentation des index avec SQL Server 2000

La commande est la suivante :

```
DBCC SHOWCONTIG ( table [ , index ] )
[ WITH { ALL_INDEXES
      | FAST [ , ALL_INDEXES ]
      | TABLERESULTS [ , { ALL_INDEXES } ]
      [ , { FAST | ALL_LEVELS } ] }
```

Notez que l'index d'id 0 est la table non cluster et l'index d'id 1 est la table sous forme de cluster. Tous les index d'id supérieur à 1 sont des index en heap.

ALL INDEXES permet de demander de voir la fragmentation de tous les index d'une table.

FAST ne recueille que les données essentielles

TABLE\_RESULTS présente les données sous forme tabulaire

ALL\_LEVELS détaille les différents niveaux des index scrutés.

Exemple :

```
DBCC SHOWCONTIG ('dbo.T1') WITH ALL_INDEXES

DBCC SHOWCONTIG analyse la table 'T1'...
Table : 'T1' (1977058079); index ID = 1, base de données ID = 58
Analyse du niveau TABLE effectuée.
- Pages analysées.....: 1111
- extensions analysées.....: 141
- extensions commutées.....: 140
- Moy des pages par extension.....: 7.9
- Densité d'analyse [meilleure valeur du compte réel].....: 98.58% [139:141]
- Fragmentation d'analyse logique..: 0.00%
- Fragmentation d'analyse d'extension..: 0.71%
- Moy octets libres par page.....: 1477.6
- Densité de page moy (pleine).....: 81.74%
DBCC SHOWCONTIG analyse la table 'T1'...
Table : 'T1' (1977058079); index ID = 2, base de données ID = 58
Analyse du niveau LEAF effectuée.
- Pages analysées.....: 1000
- extensions analysées.....: 143
- extensions commutées.....: 142
- Moy des pages par extension.....: 7.0
- Densité d'analyse [meilleure valeur du compte réel].....: 87.41% [125:143]
- Fragmentation d'analyse logique..: 0.00%
- Fragmentation d'analyse d'extension..: 0.00%
- Moy octets libres par page.....: 770.0
- Densité de page moy (pleine).....: 90.49%
```

Exécution de DBCC terminée. Si DBCC vous a adressé des messages d'erreur, contactez l'administrateur du système.

Pour voir la structure d'une page on peut aussi utiliser la commande DBCC PAGE. Cette commande ne renvoie des informations vers la console que si l'indicateur de trace 3604 a été activé.

Voici la syntaxe de DBCC PAGE :

```
DBCC PAGE ( base, fichier, page, présentation )
```

Ou présentation peut prendre une valeur allant de 1 à 3 :

1 : présentation slot à slot hexadécimale

2 : dump de page

3 : présentation slot à slot hexadécimale et en clair

### 3.2 - Voir la fragmentation des index avec SQL Server 2005

En sus du DBCC toujours utilisable sous SQL Server 2005, on peut maintenant regarder la fragmentation à l'aide de la DMV `dm_db_index_physical_stats` :

```
SELECT *
FROM sys.dm_db_index_physical_stats
(
  { database_id | NULL | 0 | DEFAULT }
  , { object_id | NULL | 0 | DEFAULT }
  , { index_id | NULL | 0 | DEFAULT }
  , { part_number | NULL | 0 | DEFAULT }
  , { LIMITED | SAMPLED | DETAILED | NULL | DEFAULT }
)
```

NULL permet de voir la fragmentation de tous les objets.

Le dernier paramètre permet d'avoir plus ou moins de détail dans la table en résultant.

## 4 - Comment éviter la fragmentation ?

Il est possible de limiter la fragmentation en utilisant différentes techniques.

En jouant sur le FILL FACTOR, c'est à dire le facteur de remplissage des pages d'index on limite la fragmentation des pages.

En utilisant exclusivement des types de données fixe pour toutes les colonnes des tables pour les colonnes qui peuvent être mise à jour, on limite la fragmentation d'extension.

De plus en utilisant des clefs d'index les plus courtes possible, par exemples des entiers auto incrémentés en lieu et place de clefs sous forme de littéraux (immatriculation, numéro de sécurité sociale...), comme en en utilisant des index monocolonne en lieu et place d'index multicolonne, on limite les effets de la fragmentation à tous niveaux.

Le FILL FACTOR permet de définir, à la création de l'index, le taux de remplissage des pages. Ainsi un FILL FACTOR de 80% permet de réserver 20% d'espace dans chaque page de données de l'index. Cela permet de minimiser notablement les splits de page. Néanmoins ce facteur de remplissage des pages d'index, s'épuise lorsque de nouvelles lignes arrivent et si l'index n'est pas reconstruit régulièrement. De plus un FILL FACTOR de 80% augmente globalement le volume de l'index de 25%.

Pour une grosse table dont les index se fragmente régulièrement, il est conseillé d'utiliser un FILL FACTOR d'au moins 90%.

Notez que le FILL FACTOR ne concerne que les pages de données. Il ne concerne pas les pages de navigation de l'index. Si l'on veut reporter ce facteur aussi aux pages de navigation, il suffit d'ajouter le mot clef PAD\_INDEX.

## 4 - Comment défragmenter les index ?

Il existe différentes commandes permettant de résorber la fragmentation des index. En fait deux manières s'affrontent : la manière agressive, qui consiste à reconstruire l'index et la manière douce qui consiste à en remanier les pages (défragmentation)..

### 4.1 - Défragmentation

En version 2000 on utilise la commande :

```
DBCC INDEXDEFRAG ( base, table, index ) [ WITH NO_INFOMSGS ]
```

Depuis la version 2005 il est plus pratique d'utiliser la commande :

```
ALTER INDEX index ON [base.][schéma.]table REORGANIZE
```

Le résultat de cette commande n'est pas optimal. Il ne livre pas un index parfaitement défragmenté. Cela s'explique par le fait que l'index continu de pouvoir être utilisé pendant la phase de la défragmentation.

C'est pourquoi on qualifie cette méthode de non agressive car l'index continuant d'être vu et pouvant être utilisé par des accès aux tables, il n'y a ni changement de plan ni allongement des délais des traitements.

### 4.2 - Reconstruction

En version 2000 on utilise les méthodes suivantes :

Suppression puis reconstruction :

```
DROP INDEX [base.][schéma.]table.index;  
CREATE INDEX index ON [base.][schéma.]table;
```

Création avec destruction :

```
CREATE INDEX index ON [base.][schéma.]table  
WITH ... DROP EXISTING
```

Cette méthode est en particulier recommandée pour les index sous jacents aux clefs primaires et contraintes d'unicité, y compris pour déplacer une table.

Reconstruction globale des index :

```
DBCC DBREINDEX ( table [ , index [ , fillfactor ] ] )
[ WITH NO_INFOMSGS ]
```

Sans spécification de l'index ce sont tous les index de la table qui sont reconstruit.

```
ALTER INDEX { index | ALL }
ON [base.][schéma.]table
{ REBUILD
  [ [ WITH ( <rebuild_index_option> [ ,...n ] ) ]
    | [ PARTITION = partition_number
      [ WITH ( <single_partition_rebuild_index_option>
        [ ,...n ] )
      ]
    ]
  ]
}
-- avec :

<rebuild_index_option > ::=
{
  PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | ONLINE = { ON | OFF }
}

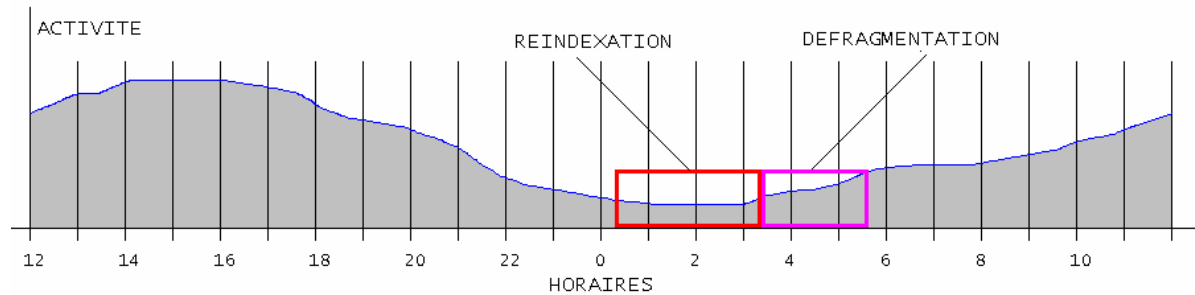
<single_partition_rebuild_index_option> ::=
{
  SORT_IN_TEMPDB = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
}
```

ONLINE = ON, permet de faire en sorte que l'index ne soit pas supprimé préalablement au tri. Au final aura lieu une substitution entre le nouvel index et l'ancien. Mais ceci impose le double de place et n'est disponible que pour l'édition Enterprise.

## 5 - Procédure pour les VLDB

Voici maintenant une procédure de maintenance d'index utilisable pour les VLDB. Son principe est décrit ci dessous.

On constate souvent que les VLDB travaillent en permanence ne livrant que peu de temps dans lequel la charge de travail baisse. C'est au moment ou la charge est la plus basse qu'il convient d'effectuer la maintenance des index. Au moment ou la charge comence à baisser sensiblement on peut opter pour un technique agressive (reconstruction). Une fois qu'elle commence à remonter il faut passer à une technique plus *soft*, jusqu'à ce que la charge redevienne normale.



La procédure dont le code figure ci dessous analyse la fragmentation et liste les index a remanier. Mais elle les présente dans un ordre aléatoire. Ceci est du au fait que la défragmentation doit inclure tous les index candidats, mais comme la fenêtre de temps de travail peut s'avérer trop courte pour ce faire, on tire au hasard les index à traiter. Ainsi en jouant cette procédure une fois par période de 24h, on est statistiquement sur que tous les index auront été traités au bout d'un certain temps.

Une approche naïve aurait été de défragmenter en priorité les index les plus fragmentés ou les plus volumineux. Mais qui dit que ce sont ceux là qui sont les plus utilisés et par conséquent ceux qui font perdre le plus de performances ?

Cette procédure permet aussi de recalculer les statistiques lors de la défragmentation si on le souhaite. Cela n'est pas nécessaire pour la reconstruction car les recalculs de statistiques sont opérés en même temps que la reconstruction.

Enfin, cette procédure utilise différentes procédures de manipulation de fichiers texte permettant de tracer les opérations effectuées.

Les paramètres de lancement de cette procédure sont les suivants :

Nom / Type	Explication
@TEMPS_MINUTE_LIMITE_REINDEX type : INT	temps limite de début de la dernière opération de réindexation, doit être > 0
@TEMPS_MINUTE_LIMITE_DEFRAG type : INT	temps limite de début de la dernière opération de défragmentation, doit être > 0
@UPDATE_STATS type : BIT, défaut 1	reconstruction des statistiques pour la phase de défragmentation : 0 pas demandé, 1, demandée
@FILE_PATH type : NVARCHAR(200)	emplacement du fichier de trace d'exécution de la défragmentation. Si NULL pas de fichier
@RATIO_EXTENTSWITCHES type : FLOAT, défaut 70	Déclenche la défragmentation si $[\text{EXTENTS} / (\text{EXTENTSWITCHES} + 1)] \% < @\text{RATIO\_EXTENTSWITCHES}$ , doit être compris entre 0 et 100
@RATIO_COUNT type : FLOAT, défaut 70	Déclenche la défragmentation si $[\text{BESTCOUNT} / \text{ACTUALCOUNT}] \% < @\text{RATIO\_COUNT}$ , doit être compris entre 0 et 100
@RATIO_DENSITY type : FLOAT, défaut 70	Déclenche la défragmentation si $[\text{SCANDENSITY}] \% < @\text{RATIO\_DENSITY}$ , doit être compris entre 0 et 100
@RATIO_FRAG type : FLOAT, défaut 70	Déclenche la défragmentation si $100 - \text{LOGICALFRAG} < @\text{RATIO\_FRAG}$ , doit être compris entre 0 et 100

En imaginant que la procédure est planifiée toutes les nuits à 22h, mettre les paramètres suivants :

- @TEMPS\_MINUTE\_LIMITE\_REINDEX = 540
- @TEMPS\_MINUTE\_LIMITE\_DEFRAG = 150

ferait que la phase de reconstruction durerait de 22h à 7h du matin et la phase de défragmentation de 7h à 9h30.

Le paramètre UPDATE\_STATS précise s'il faut recalculer les statistiques lors de la phase de défragmentation.

Le paramètre @FILE\_PATH est le nom du fichier texte (et son chemin) pour tracer les opérations effectuées. En son absence, aucune trace n'est effectuée.

Les ratios de qualité d'index ont été calculés pour cadrer entre 0 et 100 en pourcentage. À 100% il n'y a aucune fragmentation. Les seuils de déclenchement par défaut sont à 70%. Il n'est pas conseillé de les augmenter. En effet de petits index peuvent être vus comme s'ils étaient fragmentés alors que cela n'est pas forcément vrai. On peut en revanche les diminuer. En effet le critère global de retenue d'un index pour son remaniement étant un ou inclusif, à 70% sur chacun des ratios, il suffit d'un seul ratio sous cette barre pour que l'index soit candidat à la maintenance.

```

CREATE PROCEDURE dbo.P_A_MAINTINDEX_AUTO

/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot
* (http://sqlpro.developpez.com - http://www.sqlspot.com)
* Auteur : Frédéric Brouard / SQL pro
* Créée le : 2006-02-01
* Description : REINDEXATION OU DEFRAGMENTATION AUTOMATIQUE DES INDEX
* PAR SEUIL EN ORDRE ALÉATOIRE
*****/

-- temps limite d'exécution de la maintenance d'index :
@TEMPS_MINUTE LIMITE_REINDEX INT,
-- doit être > 0 : temps limite de début de la dernière opération de réindexation
@TEMPS_MINUTE LIMITE_DEFRAG INT,
-- doit être > 0 : temps limite de début de la dernière opération de défragmentation

-- pilotage du fonctionnement :
@UPDATE_STATS BIT = 1,
-- reconstruction des statistiques : 0 pas demandé, 1, demandée
@FILE_PATH NVARCHAR(200),
-- emplacement du fichier de trace d'exécution de la défragmentation.
-- Si NULL => pas de fichier.

-- seuils de déclenchement de la maintenance d'index en %. Valeur du seuil par défaut 70 :
@RATIO_EXTENTSWITCHES FLOAT = 70,
-- Déclenche la défragmentation si [EXTENTS / (EXTENTSWITCHES + 1)]% < @RATIO_EXTENTSWITCHES
@RATIO_COUNT FLOAT = 70,
-- Déclenche la défragmentation si [BESTCOUNT / ACTUALCOUNT]% < @RATIO_COUNT
@RATIO_DENSITY FLOAT = 70,
-- Déclenche la défragmentation si [SCANDENSITY]% < @RATIO_DENSITY
@RATIO_FRAG FLOAT = 70,
-- Déclenche la défragmentation si 100 - LOGICALFRAG < @RATIO_FRAG

-- exemple de lancement : EXEC dbo.P_A_MAINTINDEX_AUTO 120, 60, 1, 'C:\temp\'
AS

-- cas trivial : rien à faire !
IF @TEMPS_MINUTE LIMITE_REINDEX IS NULL OR @TEMPS_MINUTE LIMITE_DEFRAG IS NULL
RETURN

-- cas limite : rien à faire
IF @TEMPS_MINUTE LIMITE_DEFRAG < 0 OR @TEMPS_MINUTE LIMITE_REINDEX < 0
RETURN

-- les ratio doivent être en % (entre 0 et 100)
IF NOT(
    @RATIO_EXTENTSWITCHES BETWEEN 0.0 AND 100.0
    AND @RATIO_COUNT BETWEEN 0.0 AND 100.0
    AND @RATIO_DENSITY BETWEEN 0.0 AND 100.0
    AND @RATIO_FRAG BETWEEN 0.0 AND 100.0)
RETURN

```

```
PRINT 'OK3' --###

DECLARE @FILENAME VARCHAR(255)

-- vérification rapide validité du chemin
IF @FILE_PATH IS NOT NULL
BEGIN
    IF LEN(@FILE_PATH) < 2 SET @FILE_PATH = NULL
    IF SUBSTRING(@FILE_PATH, 2, 1) <> ':' SET @FILE_PATH = NULL
    IF SUBSTRING(@FILE_PATH, LEN(@FILE_PATH), 1) <> '\' SET @FILE_PATH = @FILE_PATH + '\'
    SET @FILENAME = @FILE_PATH + 'MAINTINDEX_' + REPLACE(CONVERT(CHAR(19), CURRENT_TIMESTAMP,
121), ':', '-') + '.txt'
END

-- variables locales
DECLARE @TABLE_NAME sysname
DECLARE @QUERY NVARCHAR (800)
DECLARE @SQL NVARCHAR (800)
DECLARE @OBJECTID INT
DECLARE @INDEXID INT
DECLARE @OBJECTNAME sysname
DECLARE @COMPONAME NVARCHAR(257)
DECLARE @INDEXNAME sysname
DECLARE @PAGES int
DECLARE @GUID uniqueidentifier
DECLARE @ACTUALTIME CHAR(5)
DECLARE @T_BEFORE DATETIME
DECLARE @T_AFTER DATETIME
DECLARE @T_CALC FLOAT
DECLARE @T_HMS VARCHAR(32)
DECLARE @DATABASE sysname
DECLARE @SCHEMA_NAME sysname
DECLARE @TYPEDFRG CHAR(1)
DECLARE @FS INT
DECLARE @FI INT
DECLARE @MSG NVARCHAR(800)

SET NOCOUNT ON

SET @DATABASE = DB_NAME(DB_ID())

-- Curseur sur les tables d'exploitation à l'exclusion des tables systèmes et des vues
DECLARE C_TABLES CURSOR LOCAL
FOR
    SELECT TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_TYPE = 'BASE TABLE'
    AND TABLE_NAME NOT IN (
'sysusers',
'systypes',
'syssubscriptions',
'sysreferences',
'syspublications',
'sysprotects',
'syspermissions',
'sysobjects',
'sysmergesubsetfilters',
'sysmergesubscriptions',
'sysmergeschemachange',
'sysmergepublications',
'sysmergearticles',
'sysmembers',
'sysindexkeys',
'sysindexes',
'sysfulltextcatalogs',
'sysforeignkeys',
'sysfiles',
'sysfilegroups',
'sysdepends',
'sysconstraints',
'syscomments',
'syscolumns',
'sysarticleupdates',
'sysarticles',
'Mssubscriptions',
```

```
'MSsubscription_properties',
'MSsubscriber_schedule',
'MSsubscriber_info',
'MSsnapshot_history',
'MSsnapshot_agents',
'MSreplication_subscriptions',
'MSreplication_objects',
'MSrepl_version',
'MSrepl_transactions',
'MSrepl_originators',
'MSrepl_errors',
'MSrepl_commands',
'Mspublisher_databases',
'Mspublications',
'MSpublication_access',
'MSmerge_tombstone',
'MSmerge_subscriptions',
'MSmerge_replinfo',
'MSmerge_history',
'MSmerge_genhistory',
'MSmerge_delete_conflicts',
'MSmerge_contents',
'MSmerge_agents',
'MSlogreader_history',
'MSlogreader_agents',
'MSdistributor',
'MSdistributiondbs',
'MSdistribution_history',
'MSdistribution_agents',
'MSdistpublishers',
'MSarticles',
'MSagent_profiles',
'MSagent_parameters'
)

-- Creation de la table des informations de défragmentation
CREATE TABLE #T_INDEXFRAG (
    OBJECTNAME          CHAR (255),
    OBJECTID            INT,
    INDEXNAME           CHAR (255),
    INDEXID             INT,
    LVL                 INT,
    COUNTPAGES         INT,
    COUNTROWS          INT,
    MINRECSIZE         INT,
    MAXRECSIZE         INT,
    AVGRECSIZE         INT,
    FORRECCOUNT        INT,
    EXTENTS             INT,
    EXTENTSWITCHES     INT,
    AVGFREEBYTES       INT,
    AVGPAGEDENSITY     INT,
    SCANDENSITY         DECIMAL,
    BESTCOUNT         INT,
    ACTUALCOUNT       INT,
    LOGICALFRAG        DECIMAL,
    EXTENTFRAG         DECIMAL)

-- Ouverture du curseur pour balayer toutes les tables
OPEN C_TABLES

-- Lecture première lignes
FETCH NEXT FROM C_TABLES INTO @TABLE_NAME

-- Boucle de balayage
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Execute DBCC SHOWCONTIG sur chaque table visée
    INSERT INTO #T_INDEXFRAG
    EXEC ('DBCC SHOWCONTIG ('' + @TABLE_NAME + ''')
        WITH TABLERESULTS, ALL_INDEXES, NO_INFOMSGS')

    FETCH NEXT FROM C_TABLES INTO @TABLE_NAME
END

-- Fermeture du curseur et désallocation de l'espace mémoire
```

```

CLOSE C_TABLES
DEALLOCATE C_TABLES

-- ouverture du fichier de texte pour trace de defrag
IF @FILENAME IS NOT NULL
    EXEC P_A_FILE_OPEN_REWRITE @FILENAME, @FS OUTPUT, @FI OUTPUT

-- ouverture du curseur sur les index fragmentés d'après les critères fournis, dont la volume
est de plus d'une extent (8 pages)
DECLARE C_INDEX CURSOR LOCAL
FOR
SELECT NEWID() as GUID, OBJECTID, INDEXID, OBJECTNAME, INDEXNAME, COUNTPAGES
FROM #T_INDEXFRAG
WHERE COUNTPAGES > 8
    AND ((CAST(EXTENTS AS FLOAT) / NULLIF((CAST(EXTENTSWITCHES AS FLOAT) + 1), 0)) * 100
        <= @RATIO_EXTENTSWITCHES
    OR (CAST(BESTCOUNT AS FLOAT) / NULLIF(CAST(ACTUALCOUNT AS FLOAT), 0)) * 100
        <= @RATIO_COUNT
    OR CAST(SCANDENSITY AS FLOAT)
        <= @RATIO_DENSITY
    OR 100 - CAST(LOGICALFRAG AS FLOAT)
        <= @RATIO_FRAG
    )
ORDER BY GUID

OPEN C_INDEX

-- lecture du premier index à défragmenter
FETCH NEXT FROM C_INDEX INTO @GUID, @OBJECTID, @INDEXID, @OBJECTNAME, @INDEXNAME, @PAGES

SET @TYPEDFRG = 'R'

-- boucle sur les index
WHILE @@FETCH_STATUS = 0
BEGIN
    SET @QUERY = NULL
    SET @OBJECTNAME = RTRIM(@OBJECTNAME)
    SET @INDEXNAME = RTRIM(@INDEXNAME)
    -- récupération du 'propriétaire' de l'objet (en fait le schéma)
    SELECT @SCHEMA_NAME = name
    FROM sysusers
    WHERE uid = OBJECTPROPERTY(@OBJECTID, 'OwnerId')
    IF @@ERROR <> 0 GOTO LBL_RESUME
    SET @COMPONAME = @SCHEMA_NAME + '.' + @OBJECTNAME
    SET @T_BEFORE = CURRENT_TIMESTAMP
    -- capture de l'heure avant l'exécution
    IF CURRENT_TIMESTAMP >= DATEADD(m, @TEMPS_MINUTE LIMITE REINDEX
        + @TEMPS_MINUTE LIMITE DEFRAG, @T_BEFORE)

    -- c'est déjà trop tard !
    GOTO LBL_RESUME
    IF CURRENT_TIMESTAMP >= DATEADD(m, @TEMPS_MINUTE LIMITE REINDEX, @T_BEFORE)
    -- on est passé à la défragmentation
    BEGIN
        SET @TYPEDFRG = 'D'
        IF RTRIM(@INDEXNAME) <> ''
        BEGIN
            SELECT @QUERY = 'DBCC INDEXDEFRAG (' + @DATABASE + ', ' + @COMPONAME + ', '
                + @INDEXNAME + ') WITH NO_INFOMSGS'

            EXEC (@QUERY)
            IF @@ERROR <> 0 GOTO LBL_ERROR
            IF @UPDATE_STATS = 1
            BEGIN
                -- il faut mettre à jour les stats. Sur la table entière si moins de 80 Mo,
                -- 50% si moins de 320 Mo, 25% si moins de 1,3 Go Mo, 12 % si moins de 7,8 Go,
                -- 8% au delà
                SET @SQL = 'UPDATE STATISTICS [' + @SCHEMA_NAME + '].[' + @OBJECTNAME + ']' ('
                    + @INDEXNAME + ') '
                SET @SQL = @SQL + CASE
                    WHEN @PAGES > 640000 THEN ' SAMPLE 8 PERCENT'
                    WHEN @PAGES > 160000 THEN ' SAMPLE 12 PERCENT'
                    WHEN @PAGES > 40000 THEN ' SAMPLE 25 PERCENT'
                    WHEN @PAGES > 10000 THEN ' SAMPLE 50 PERCENT'
                END
                EXEC (@SQL)
            END
        END
    END
END
END
END

```

```

END
ELSE
BEGIN
-- on reconstruit les indexs
IF RTRIM(@INDEXNAME) = ''
-- c'est la table entière
SELECT @QUERY = 'DBCC DBREINDEX (''[' + @DATABASE + '].[' + @SCHEMA_NAME + '].['
+ RTRIM(@OBJECTNAME) + ']'', ''', 0)'

ELSE
-- c'est juste un index
SELECT @QUERY = 'DBCC DBREINDEX (''[' + @DATABASE + '].[' + @SCHEMA_NAME + '].['
+ RTRIM(@OBJECTNAME) + ']'', '' + @INDEXNAME
+ ''', 0)'

EXEC (@QUERY)
IF @@ERROR <> 0 GOTO LBL_ERROR
END
-- calcul du temps d'exécution
SET @T_AFTER = CURRENT_TIMESTAMP
SET @T_CALC = (CAST(@T_AFTER AS FLOAT) - CAST(@T_BEFORE AS FLOAT)) * 24
SET @T_HMS = CAST(CAST(FLOOR(@T_CALC) AS INT) AS VARCHAR(32)) + 'h'
SET @T_CALC = (@T_CALC - FLOOR(@T_CALC)) * 60
SET @T_HMS = @T_HMS + ' ' + CAST(CAST(FLOOR(@T_CALC) AS INT) AS VARCHAR(32)) + 'm'
SET @T_CALC = (@T_CALC - FLOOR(@T_CALC)) * 60
SET @T_HMS = @T_HMS + ' ' + CAST(CAST(FLOOR(@T_CALC) AS INT) AS VARCHAR(32)) + 's'
-- écriture de la passe de defrag dans fichier de trace
IF @FILENAME IS NOT NULL
BEGIN
EXEC P_A_FILE_WRITE_LINE @FI, @QUERY
IF @SQL IS NOT NULL
EXEC P_A_FILE_WRITE_LINE @FI, @SQL
IF @TYPEDFRG = 'D'
SET @MSG = 'Défragmentation '
IF @TYPEDFRG = 'R'
SET @MSG = 'Réindexation '
SET @MSG = @MSG + 'effectuée dans la base ' + @DATABASE +
' pour l''index ' + @INDEXNAME + ' de l''objet '
+ @SCHEMA_NAME + '.' + @OBJECTNAME +
' en ' + @T_HMS + ' à ' + CONVERT(VARCHAR(24), @T_BEFORE, 121)
EXEC P_A_FILE_WRITE_LINE @FI, @MSG
END

FETCH NEXT FROM C_INDEX INTO @GUID, @OBJECTID, @INDEXID, @OBJECTNAME, @INDEXNAME, @PAGES
END

GOTO LBL_RESUME

LBL_ERROR:

-- inscrire l'erreur dans le fichier de trace
SET @QUERY = 'ERREUR : ' + @QUERY
IF @FILENAME IS NOT NULL
EXEC P_A_FILE_WRITE_LINE @FI, @QUERY

LBL_RESUME:

-- le curseur est-ile encore ouvert ?
IF CURSOR_STATUS ('local', 'C_INDEX') > 0
BEGIN
CLOSE C_INDEX
DEALLOCATE C_INDEX
END

DROP TABLE #T_INDEXFRAG;

-- fermeture du fichier de trace
EXEC P_A_FILE_CLOSE @FS, @FI

GO

```

Notez que cette procédure utilise plusieurs autres procédures de gestion des fichiers textes dont voici le code :

```

/*
routines de gestion de fichiers texte

```

```
DROP PROCEDURE P_A_FILE_OPEN_REWRITE
DROP PROCEDURE P_A_FILE_OPEN_APPEND
DROP PROCEDURE P_A_FILE_WRITE_LINE
DROP PROCEDURE P_A_FILE_WRITE_DATA
DROP PROCEDURE P_A_FILE_CLOSE
*/

CREATE PROCEDURE P_A_FILE_OPEN_REWRITE @FILE_NAME NVARCHAR(256), -- nom du fichier
                                       @F_HANDLE INT OUTPUT, -- handle OS du fichier
                                       @F_ID INT OUTPUT -- identifiant numérique du
fichier
AS
/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot *
* (http://sqlpro.developpez.com - http://www.sqlspot.com) *
* Auteur : Frédéric Brouard / SQL pro *
* Créée le : 2006-02-01 *
* Description : ouvre un fichier texte en écriture à l'aide d'objets OLE *
*****/

BEGIN

    SET NOCOUNT ON

    -- valeur de retour d'exécution appel OLE
    DECLARE @OLE_RETURN INT

    -- création de l'objet
    EXECUTE @OLE_RETURN = sp_OACreate 'Scripting.FileSystemObject', @F_HANDLE OUT
    IF @OLE_RETURN <> 0 RETURN -1

    --Ouvre le fichier (2 = ForWriting, 8 = ForAppending)
    EXECUTE @OLE_RETURN = sp_OAMethod @F_HANDLE, 'OpenTextFile', @F_ID OUT, @FILE_NAME, 2, 1
    IF @OLE_RETURN <> 0 RETURN -1

END
GO

CREATE PROCEDURE P_A_FILE_OPEN_APPEND @FILE_NAME NVARCHAR(256), -- nom du fichier
                                       @F_HANDLE INT OUTPUT, -- handle OS du fichier
                                       @F_ID INT OUTPUT -- identifiant numérique du
fichier
AS
/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot *
* (http://sqlpro.developpez.com - http://www.sqlspot.com) *
* Auteur : Frédéric Brouard / SQL pro *
* Créée le : 2006-02-01 *
* Description : ouvre un fichier texte en ajout à l'aide d'objets OLE *
*****/

BEGIN

    SET NOCOUNT ON

    -- valeur de retour d'exécution appel OLE
    DECLARE @OLE_RETURN INT

    -- création de l'objet
    EXECUTE @OLE_RETURN = sp_OACreate 'Scripting.FileSystemObject', @F_HANDLE OUT
    IF @OLE_RETURN <> 0 RETURN -1

    --Ouvre le fichier (2 = ForWriting, 8 = ForAppending)
    EXECUTE @OLE_RETURN = sp_OAMethod @F_HANDLE, 'OpenTextFile', @F_ID OUT, @FILE_NAME, 8, 1
    IF @OLE_RETURN <> 0 RETURN -1

END
GO

CREATE PROCEDURE P_A_FILE_WRITE_LINE @F_ID INT, -- identifiant numérique du
fichier
                                       @LINE NVARCHAR(4000) -- ligne de texte à écrire
AS
```

```

/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot
              (http://sqlpro.developpez.com - http://www.sqlspot.com)
* Auteur : Frédéric Brouard / SQL pro
* Créée le : 2006-02-01
* Description : ajoute une ligne à un fichier texte l'aide d'objets OLE
*****/

BEGIN

    SET NOCOUNT ON

    -- valeur de retour d'exécution appel OLE
    DECLARE @OLE_RETURN INT

    -- ecrit la ligne dans le fichier
    EXECUTE @OLE_RETURN = sp_OAMethod @F_ID, 'WriteLine', Null, @LINE

    IF @OLE_RETURN <> 0 RETURN -1

END
GO

CREATE PROCEDURE P_A_FILE_WRITE_DATA @F_ID INT, -- identifiant numérique du
fichier                                     @LINE NVARCHAR(4000) -- texte à ajouter
AS

/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot
              (http://sqlpro.developpez.com - http://www.sqlspot.com)
* Auteur : Frédéric Brouard / SQL pro
* Créée le : 2006-02-01
* Description : ajoute du texte à un fichier texte l'aide d'objets OLE
*****/

BEGIN

    SET NOCOUNT ON

    -- valeur de retour d'exécution appel OLE
    DECLARE @OLE_RETURN INT

    -- ecrit la ligne dans le fichier
    EXECUTE @OLE_RETURN = sp_OAMethod @F_ID, 'Write', Null, @LINE

    IF @OLE_RETURN <> 0 RETURN -1

END
GO

CREATE PROCEDURE P_A_FILE_CLOSE @F_HANDLE INT, -- handle OS du fichier
                                @F_ID int -- identifiant numérique du fichier
AS

/*****
* Copyright : Frédéric Brouard / SQLpro / SQL spot
              (http://sqlpro.developpez.com - http://www.sqlspot.com)
* Auteur : Frédéric Brouard / SQL pro
* Créée le : 2006-02-01
* Description : ferme un fichier texte l'aide d'objets OLE
*****/

BEGIN

    SET NOCOUNT ON

    DECLARE @OLE_RETURN int

    -- fermeture du fichier
    EXECUTE @OLE_RETURN = sp_OADestroy @F_ID

    -- destruction de l'objet Ole
    EXECUTE @OLE_RETURN = sp_OADestroy @F_HANDLE


```

```
END  
GO
```

Si vous les utilisez sous MS SQL Server 2008, vous aurez besoins d'activer l'usage des procédure OLE à l'aide des commandes :

```
sp_configure 'show advanced options', 1;  
GO  
RECONFIGURE;  
GO  
sp_configure 'Ole Automation Procedures', 1;  
GO  
RECONFIGURE;  
GO
```