

Volume

1

HELA – CATEGORIE ECONOMIQUE - ISAT

Section Informatique de Gestion – 2^{ème} Année

SQL 2008 Serveur

Volume1 : Création et Mise en service d'une base de Donnée SQL 2008
Serveur



Introduction

Rappels sur le stockage des données

Le stockage des données représente un problème aussi vieux que l'informatique. Au fur et à mesure de l'évolution des capacités techniques du matériel et du volume des données manipulées, la façon de stocker et d'organiser les données a lui aussi évolué.

Dans le cadre d'une application de gestion, toutes les catégories de données ne sont pas concernées de la même façon par ces problèmes d'organisation.

1. Les différentes catégories de données

Dans un premier temps, il convient de définir la catégorie des données. Cette catégorisation est issue de quelques questions simples :

- À quoi servent les données ?
- Combien de temps est-il nécessaire de conserver ces données ?

a. Les données de base

Ce type de données est au cœur de tout système d'information. Il s'agit des données à partir desquelles il est possible de travailler. Ainsi, dans le cadre d'une gestion commerciale, les données de bases seront les informations sur les clients et sur les produits. Les données de ce type sont aussi volumineuses que possible et bien entendu elles ont une durée de vie très longue. Comme ce sont des données de base, elles devront être accessibles facilement et rapidement.

b. Les données de mouvement

Ces données sont générées à partir des données de base. Contrairement à ces dernières, leur durée de vie sera limitée mais leur volume sera beaucoup plus important. Par exemple, toujours dans le cadre d'une gestion commerciale, les informations relatives à chaque commande sont considérées comme des données de mouvement. Le volume est important car l'entreprise compte bien que chaque client passe plusieurs commandes au cours d'une



même année comptable. Par contre, la durée de vie de ces informations est bien moindre. En effet, il n'est pas nécessaire de conserver ce type d'informations plusieurs années en ligne, mais plutôt sur un support d'archivage autre et moins coûteux.

c. Les données de travail

Il s'agit de données générées dans un but précis, avec un volume parfois important mais une durée de vie très courte. Dès que le travail est réalisé il n'est pas nécessaire de conserver ces données. Ainsi, par exemple, les données extraites de la base et qui vont servir à la réalisation de graphiques sont à ranger dans cette catégorie. Dès que les graphiques sont réalisés, il n'est plus nécessaire de conserver les données extraites de la base qui ont permis de les obtenir.

d. Les données d'archive

Il s'agit de données très volumineuses et avec une durée de vie très longue mais qui présentent la caractéristique de ne pas être directement accessibles. Lorsqu'elles le sont, c'est uniquement en lecture. Par exemple, dans le cadre d'une application de gestion commerciale, il peut s'agir des données relatives aux années comptables passées.

2. L'organisation des données

a. Directe

Cette organisation est sans doute la plus simple à utiliser. Les données sont enregistrées les unes à la suite des autres dans un fichier. Chaque ensemble de données possède une longueur fixe et les enregistrements sont stockés les uns derrière les autres. Ainsi, la connaissance de la longueur d'un enregistrement permet par simple calcul d'accéder directement au 10^{ème} enregistrement.

Ce type d'organisation est coûteux en espace disque et ne permet pas d'extraire facilement les informations sur des critères autres que leur position dans l'ordre d'enregistrement.

b. Séquentielle



Avec l'organisation séquentielle les données sont enregistrées les unes à la suite des autres. Un caractère spécial est utilisé pour marquer la séparation entre les différents champs tandis qu'un autre est utilisé pour marquer la fin de chaque enregistrement. Les caractères retenus sont couramment la virgule (,) et la fin de ligne (CR). Les fichiers qui retiennent ces séparateurs sont alors décrits comme des fichiers CSV (*Comma Separated Values*).

Ce type d'organisation permet d'optimiser l'espace de stockage utilisé et résoud ainsi l'un des problèmes majeurs des fichiers avec un accès direct. Par contre, comme pour l'organisation directe, lorsque l'on recherche des données répondant à des critères de sélection bien précis il est nécessaire de parcourir l'ensemble des données, ce qui s'avère d'autant plus long que le volume de données (nombre d'enregistrements) est important.

c. Séquentielle indexée

Les données sont toujours stockées au format séquentiel mais afin de permettre un accès plus rapide aux informations, des index peuvent être définis pour chaque fichier. À l'intérieur de ces index les données sont triées par ordre alphanumérique. Le parcours de l'index est réalisé de façon séquentielle et permet un accès direct aux informations stockées dans le fichier de données.

Le parcours de l'index, bien que séquentiel, est rapide car le volume de données manipulé est faible. De plus, comme les données sont triées, il n'est pas nécessaire de lire la totalité de l'index. Enfin, il est possible de définir plusieurs index sur un même fichier de données. Par exemple, sur un fichier stockant des informations relatives aux clients, il est possible de définir un index sur les noms et un autre sur les villes.

Avec ce type d'organisation, la difficulté consiste à maintenir à jour les index lors des opérations d'ajout, de suppression et de mise à jour. De plus, comme avec les organisations directe et séquentielle, les fichiers ne sont pas liés les uns aux autres et il n'existe pas de contexte de sécurité au niveau des données. Par exemple, rien ne s'oppose, au niveau des données, à la suppression d'un client même s'il possède des commandes en cours.

De même, toute personne en mesure de travailler avec les données, peut accéder à la totalité des données en lecture et en écriture. Ces inconvénients posent plus de problèmes avec l'organisation séquentielle indexée car des volumes de données important peuvent être gérés ainsi avec de nombreux utilisateurs connectés.

Cette solution séquentielle indexée a été adoptée de façon massive pour des applications petites à moyennes car afin de faciliter les développements, de nombreux langages de programmation proposaient un moteur de gestion de ce type d'organisation.



d. Base de données hiérarchique

Avec ces bases de données, les problèmes de sécurité d'accès aux données ainsi que la liaison entre les données ont été résolus. Par contre, chaque moteur a été développé de façon indépendante par les différents éditeurs.

L'apprentissage du moteur est donc à recommencer à chaque fois que l'on développe avec un nouveau moteur (le langage d'interrogation, l'API d'accès aux données). Ce à quoi il faut ajouter une organisation complexe des données.

Ces solutions hautement propriétaires sont souvent très coûteuses pour l'entreprise qui les choisit.

e. Base de données relationnelle

Fondée sur une représentation logique des données en respectant le modèle relationnel, les bases de données relationnelles ont su s'imposer car elles s'appuient toutes sur le même langage standardisé et normalisé qu'est le SQL.

3. La normalisation du schéma relationnel

Lorsque le schéma relationnel est défini afin de répondre à tous les besoins des utilisateurs, il est nécessaire de le normaliser afin d'éviter toute redondance d'information ainsi que toute structure non conforme avec le modèle relationnel.

Lorsque cette opération est réalisée, le schéma pourra alors être dénormalisé bien que cette opération soit rarement la meilleure. Si le développeur dénormalise le schéma, il doit également mettre en place l'ensemble du mécanisme qui permet de maintenir la cohérence des données. En effet, le modèle relationnel, et donc les SGBDR (Système de Gestion de Base de données Relationnelle), ne peuvent garantir la cohérence des données que sur des modèles normalisés.

Les formes normales permettent de s'assurer que le schéma est bien conforme au modèle relationnel. Il existe de façon théorique cinq formes normales, mais dans la pratique, seules les trois premières sont appliquées. L'application des formes normales nécessite de bien maîtriser le concept de dépendance fonctionnelle. Une donnée dépend fonctionnellement d'une autre lorsque la connaissance de la seconde permet de déterminer la valeur de la première. Par exemple, il est possible de dire que dans une application de gestion commerciale, il existe une dépendance fonctionnelle entre un code TVA et le taux de TVA ou bien entre la référence d'un article et sa désignation.



Première forme normale : une table est dite en première forme normale lorsque toutes les colonnes contiennent des valeurs simples.

Par exemple, si une table des clients contient un champ Telephones dans lequel les différents numéros de téléphone d'un client sont stockés, alors cette table n'est pas en première forme normale. Il est alors nécessaire de définir les colonnes Bureau et Mobile afin de mieux structurer les données.

Clients	Numero	Nom	Prenom	Telephones	Commande	Du
	1	DUPONT	Jean	01 02 03 04 05 06 07 08 09 10	1350	01/01/2008
	1	DUPONT	Jean	01 02 03 04 05 06 07 08 09 10	1352	15/01/2008
	2	DURAND	Pauline	01 03 05 07 09	1351	02/01/2008

La table présentée ci dessus ne respecte pas la première forme normale.

Cette table respecte la première forme normale.

Clients	Numero	Nom	Prenom	Bureau	Mobile	Commande	Du
	1	DUPONT	Jean	01 02 03 04 05	06 07 08 09 10	1350	01/01/2008
	1	DUPONT	Jean	01 02 03 04 05	06 07 08 09 10	1352	15/01/2008
	2	DURAND	Pauline	01 03 05 07 09		1351	02/01/2008

Deuxième forme normale : une table est dite en deuxième forme normale si elle est en première forme normale et si toutes les colonnes non clés dépendent fonctionnellement de la clé primaire.

En reprenant l'exemple présenté ci dessus, il est possible d'admettre dans un premier temps que la clé de la table des clients est composée des colonnes Numero et Commande.

Dans ce cas, les valeurs des colonnes Nom, Prenom, Bureau et Mobile dépendent uniquement du numéro tandis que la colonne Du est liée au numéro de la commande. La table n'est donc pas en seconde forme normale. Il est donc nécessaire de définir deux tables : clients et commandes.

Partie 1

Client	Numero	Nom	Prenom	Bureau	Mobile
	1	DUPONT	Jean	01 02 03 04 05	06 07 08 09 10
	2	DURAND	Pauline	01 03 05 07 09	

Commande	Numero	Du	Client
	1350	01/01/2008	1
	1352	15/01/2008	1
	1351	02/01/2008	2

Les deux tables présentées ci dessus respectent la deuxième forme normale.

Troisième forme normale : une table est dite en troisième forme normale si elle est en deuxième forme normale et s'il n'existe pas de dépendance fonctionnelle entre deux colonnes non clé.

Par exemple, si dans la table des clients les colonnes Civilite et Sexe sont ajoutées de la façon suivante :

Client	Numero	Nom	Prenom	Bureau	Mobile	Civilite	Sexe
	1	DUPONT	Jean	01 02 03 04 05	06 07 08 09 10	M	M
	2	DURAND	Pauline	01 03 05 07 09		Mlle	F

Il est alors possible de dire qu'il existe une dépendance fonctionnelle entre le sexe et la civilité. En effet, le fait de connaître la civilité (Mlle, Mme ou M) permet de déduire le sexe. La table des clients ne respecte donc pas la troisième forme normale. La table des civilités est définie de façon à obtenir le schéma suivant :

Client	Numero	Nom	Prenom	Bureau	Mobile	Civilite
	1	DUPONT	Jean	01 02 03 04 05	06 07 08 09 10	M
	2	DURAND	Pauline	01 03 05 07 09		Mlle

Civilite	Valeur	Sexe
	Mlle	F
	Mme	F
	M	M

Les deux tables présentées ci dessus respectent la troisième forme normale.

Le modèle relationnel

1. Concepts et définitions

Le modèle relationnel repose sur des concepts de base simples (domaine, relation, attribut), auxquels s'appliquent des règles précises. La mise en oeuvre de la base est facilitée par un langage assertien (non procédural) simple, basé sur une logique ensembliste. C'est un ensemble de valeurs caractérisé par un nom.

Cardinal

C'est le nombre d'éléments d'un domaine.

Exemple

Le dictionnaire des données de l'analyse d'une gestion commerciale peut comporter, entre autres, des spécifications sur la gestion des états de commande ou des numéros d'ordre à afficher.

Le modèle relationnel les traduira de la manière suivante :

États des commandes = {"EC", "LI", "FA", "SO"}; cardinal 4

Numéros d'ordre = {n | 1 ≤ n ≤ 9999}; cardinal 9999.

Le produit cartésien P entre plusieurs domaines D1, D2, ..., Dn noté $P = D1 \times D2 \times \dots \times Dn$ est l'ensemble des n uplets (tuples) (d1, d2, ..., dn) où chaque di est un élément du domaine Di.

Exemple

Si on veut gérer deux domaines (codes et taux), on pourra obtenir des 2uplets composés d'un code et d'un taux.

Codes = {1,2,3,4}

Taux de TVA = {0,5.5,19.6}

Codes X Taux de TVA = {(1,0), (1,5.5), (1,19.6),
(2,0), (2,5.5), (2,19.6), (3,0), (3,5.5), (3,19.6),



$(4,0),(4,5.5),(4,19.6)\}$

Une relation définie sur les domaines D_1, D_2, \dots, D_n est un sous ensemble du produit cartésien de ces domaines caractérisé par un nom.

Attribut

C'est une colonne d'une relation caractérisée par un nom.

Degré

C'est le nombre d'attributs d'une relation.

Exemple

Pour associer un seul taux par code, seuls trois 2uplets doivent être concernés.

Relation TVA = $\{(1,0),(2,5.5),(3,19.6)\}$

Elle se fait sous forme de tableau (table), en extension :

ou en compréhension :

TVA (CODE:codes, VALEUR:Taux de TVA)

ou

TVA (CODE, VALEUR)

2 . Principales règles

Le modèle relationnel gère donc un objet principal, la relation, associée aux concepts de domaine et d'attribut. Des règles s'appliquent à cette relation afin de respecter les contraintes liées à l'analyse.

Quelques unes de ces règles sont :

- Toute valeur prise par un attribut doit appartenir au domaine sur lequel il est défini.
- Tous les éléments d'une relation doivent être distincts.
- Attribut ou ensemble d'attributs permettant de caractériser de manière unique chaque élément de la relation.
- Identifiant minimum d'une relation.
- Autres identifiants de la relation.

Partie 1

Cette règle impose qu'un attribut ou ensemble d'attributs d'une relation apparaisse comme clé primaire dans une autre relation.

Attribut ou ensemble d'attributs vérifiant la règle d'intégrité référentielle.

Exemple

L'analyse d'une gestion commerciale nous impose de gérer des clients ayant des caractéristiques (Nom, adresse) et des commandes que passent ces clients.

On pourra proposer le modèle suivant :

CLIENTS	NUMEROCLI	NOMCLI	ADRESSECLI
	15	DUPONT S.A.	NANTES
	20	Etb. LABICHE	PARIS
	35	DUBOIS Jean	NANTES
	138	DUBOIS Jean	TOURS

COMMANDES	NUMEROCDE	DATECDE	NUMEROCLI	ETATCDE
	1210	15/10/94	15	SO
	1230	18/10/94	35	SO
	1301	20/11/94	15	EC
	1280	02/11/94	20	LI
	1150	18/03/94	15	SO
	1250	02/11/94	35	EC

CLIENTS (NUMEROCLI,NOMCLI,ADRESSECLI)

NUMEROCLI identifiant clé primaire de CLIENTS

NOMCLI,ADRESSECLI identifiant clé secondaire de CLIENTS

COMMANDES (NUMEROCDE,DATECDE,NUMEROCLI,ETATCDE)

NUMEROCDE identifiant clé primaire de COMMANDES

NUMEROCLI clé étrangère de COMMANDES, référençant

NUMEROCLI de CLIENTS

Dans le modèle relationnel, la notion de nullité est admise. C'est une valeur représentant une information inconnue ou inapplicable dans une colonne. Elle est notée NULL.

Toute valeur participant à une clé primaire doit être non NULL.

Exemple

Dans la relation article, on admet que le prix ou le code TVA peuvent être inconnus, mais la référence de l'article (clé primaire) doit être renseignée.

ARTICLES	REFART	DESIGNATION	PRIX	CODETVA	CATEGORIE
	AB10	Tapis de Chine	1 500.00	2	IMPORT
	AB22	Tapis Persan	1 250.10	2	IMPORT
	CD50	Chaîne HIFI	735.40	2	IMPORT
	ZZZZ	Article bidon	NULL	NULL	DIVERS
	AA00	Cadeau	0.00	NULL	DIVERS
	AB03	Carpette	150.00	2	SOLDES
	AB	Tapis	NULL	2	DIVERS
	ZZ01	Lot de tapis	500.00	2	DIVERS

Gérer une base de données

La création et la maintenance d'une base de données SQL Server vont toucher des domaines d'activité variés, qui sont :


- la gestion de l'espace de stockage,
- la configuration de la base de données,
- la gestion des objets de la base,
- la traduction des contraintes de l'analyse,
- la gestion de la sécurité d'accès,
- les sauvegardes.
-

Certains de ces domaines touchent également l'administrateur et seront étudiés ultérieurement. La gestion et la configuration de SQL Server peuvent se faire de deux manières ; soit en Transact SQL, en interactif ou par script, soit par Microsoft SQL Server Management Studio, avec l'interface graphique.

Partie 1

Avec SQL Server 2008, il existe trois types de base de données :

- Les bases OLTP (*OnLine Transaction Processing*), c'est à dire des bases qui vont supporter les transactions des utilisateurs. C'est ce type de base qui se trouve en production. Les principales caractéristiques de ce type de base sont que, malgré un volume de données conséquent et de nombreux utilisateurs connectés, les temps de réponse doivent être optimum. Heureusement, les utilisateurs travaillent sous forme de transaction courte et chaque transaction manipule un faible volume de données.
- Les bases OLAP (*OnLine Analytical Processing*), c'est à dire une base qui va permettre de stocker un maximum d'informations afin de faire des requêtes d'aide à la prise de décision. C'est le monde du décisionnel qui n'est pas abordé dans ce syllabus.
- Les bases de type snapshot, qui sont des répliquions plus ou moins complètes de la base d'origine afin d'accéder de façon rapide à des données éloignées par exemple.

 Seule la notion de base de données de type utilisateur est abordée ici. La gestion des bases de données systèmes n'est pas évoquée.

1. Gérer l'espace de stockage

SQL Server utilise un ensemble de fichiers pour stocker l'ensemble des informations relatives à une base.

Il en existe un seul par base de données, c'est le point d'entrée. Ce fichier porte l'extension *.mdf.

Il peut en exister plusieurs par base de données. Ils portent l'extension *.ndf.

Ces fichiers (il peut y en avoir plusieurs) contiennent le journal des transactions et portent l'extension *.ldf.

Il est possible de préciser un groupe de fichiers lors de la mise en place des fichiers. Ces groupes présentent l'avantage d'équilibrer les charges de travail sur les différents disques du système. Les données sont écrites de façon équitable sur les différents fichiers du groupe.

2. Structure des fichiers de données



taille maximale d'une ligne est de 8060 octets hors types texte et image. Cette taille de 8 Ko autorise :

- de meilleurs temps de réponse lors des opérations de lecture/écriture.
- de supporter des lignes de données plus longues et donc de moins faire appel aux types texte et image.
- une meilleure gestion des bases de très grande taille.

Ces pages sont regroupées dans des extensions. Les extensions sont constituées de huit pages contiguës (64 Ko).

Elles représentent l'unité d'allocation d'espace aux tables et aux index. Pour éviter de perdre de la place disque, il existe deux types d'extensions :

- Uniforme

Réservée à un seul objet.

- Mixte

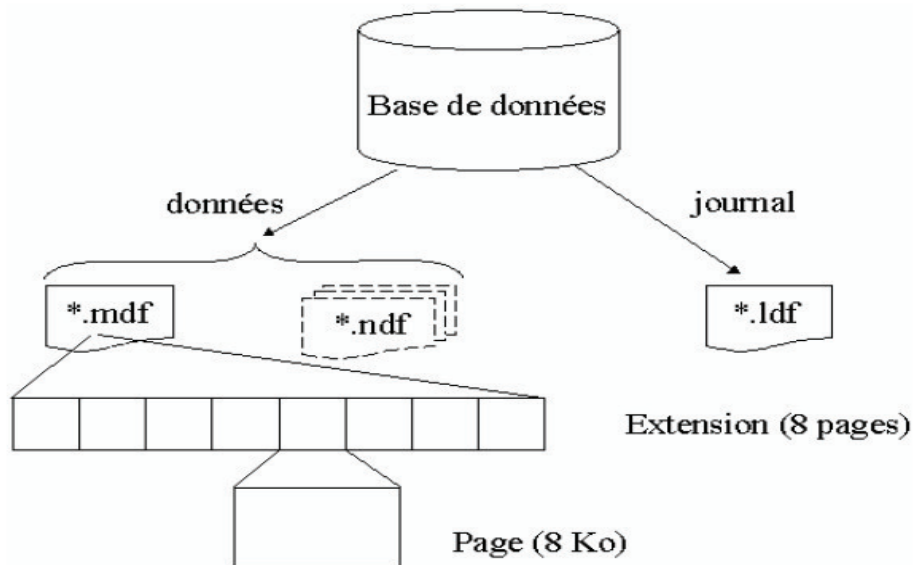
Partagée par plusieurs objets, 8 au maximum.

Lorsqu'une table est créée, les pages sont allouées dans une extension mixte. Quand les données représentent huit pages, alors une extension uniforme est allouée à la table.

3. Les fichiers de données

Les fichiers de données peuvent être redimensionnés de façon dynamique ou manuelle. Lors de la création du fichier, il faut préciser :

- Le nom logique du fichier pour le manipuler avec le langage Transact SQL.
- Le nom physique pour préciser l'emplacement du fichier.
- Une taille initiale.
- Une taille maximale.
- Un pas d'incrément.



4. Gérer l'objet DATABASE

Une DATABASE contient tous les autres objets :

- Le catalogue de base de données.
- Les objets utilisateurs (tables, defaults, views, rules, triggers, procédures).
- Les index, les types de données, les contraintes.
- Le journal de transactions.

La personne qui crée la base doit disposer des droits suffisants et devient le propriétaire de la base. SQL Server est capable de gérer 32767 bases de données.

➤ Les fichiers qui constituent une base de données ne doivent pas être placés sur un système de fichiers compressé ou un répertoire réseau partagé.

Lors de leur création, les fichiers sont initialisés avec des zéros de façon à écraser toutes les données déjà présentes.

Cette opération entraîne une surcharge de travail lors de la création des fichiers mais elle permet d'optimiser les temps de réponse lorsque la base est en production.



Il est possible d'allouer de l'espace disque à la base sans qu'il soit initialisé par des 0. Cette opération est identifiée sous le terme initialisation instantanée. Les données anciennement présentes sur le disque sont écrasées au fur et à mesure des besoins de la base.

SQL Server est capable d'utiliser des partitions brutes pour la création des fichiers de bases de données. Cependant, dans la très grande majorité des cas, la méthode à privilégier est de créer les fichiers sur une partition NTFS.

En effet, l'utilisation d'une partition brute ne permet pas de signaler au système d'exploitation, et donc à l'administrateur, que cette partition est actuellement utilisée par SQL Server. L'espace, non utilisé du point de vue système, peut ainsi être facilement utilisé pour étendre une partition ou bien créer une nouvelle partition. En utilisant des partitions brutes le risque de mauvaises manipulations augmente donc considérablement pour un gain qui n'est que peu significatif. Les fichiers créés sur des partitions NTFS supportent sans soucis la compression NTFS et éventuellement certains groupes de fichiers peuvent être positionnées en lecture seule. Ces considérations ne s'appliquent qu'aux bases de données utilisateur et ne peuvent pas être appliquées sur les fichiers relatifs aux bases de données système.

a. Créer la base

Pour créer une base de données, il faut être connecté en tant qu'administrateur système, ou avoir la permission d'utiliser CREATE DATABASE, et être dans la base de données système master.

L'objet DATABASE doit être créé en premier lieu. Une base de données contient tous les autres objets :

- Le catalogue de base de données.
- Les objets utilisateurs (tables, valeurs par défaut, vues, règles, déclencheurs, procédures).
- Les index, les types de données, les contraintes d'intégrité.
- Le journal des transactions.

Le nom de la base de données doit être unique dans une instance SQL Server. Ce nom est limité à 128 caractères en respectant les règles de construction des identificateurs. Cette longueur maximale est réduite à 123 caractères si le nom du journal n'est pas précisé lors de la création de la base.

Syntaxe

```
CREATE DATABASE nom_base [ ON [PRIMARY]
```

```
(( [ NAME = nom Logique, ]  
FILENAME = 'nom Physique'  
[, SIZE = taille]  
[, MAXSIZE = { taille Maxi | UNLIMITED } ]  
[, FILEGROWTH = valeur Incrément] ) [,...]  
[ LOG ON { fichier } ]  
[COLLATE nom_classement]  
[ FOR ATTACH | FOR ATTACH_REBUILD_LOG ]
```

NAME

Nom logique du fichier.

FILENAME

Emplacement et nom physique du fichier.

SIZE

Taille initiale du fichier en mégaoctets (MB) ou kilooctets (KB). La taille par défaut est de 1 mégaoctet.

MAXSIZE

Taille maximum du fichier indiquée en kilo ou mégaoctets (par défaut mégaoctets). Si aucune valeur n'est précisée, alors la taille du fichier sera limitée par la place libre sur le disque.

UNLIMITED

Pas de taille maximum, la limite est la place libre sur le disque.

FILEGROWTH

Précise le pas d'incrément pour la taille du fichier, qui ne pourra jamais dépasser la valeur maximale. Ce pas peut être précisé en pourcentage ou de façon statique en kilo ou mégaoctets. Les extensions possèdent une taille de 64 Ko.

C'est donc la valeur minimale du pas d'incrément qu'il faut fixer.

LOG ON

Emplacement du journal de transactions. Le journal de transactions stocke les modifications apportées aux données.

À chaque INSERT, UPDATE ou DELETE, une écriture est faite dans le journal avant l'écriture dans la base. La validation des transactions est également consignée dans le journal. Ce journal sert à la récupération des données en cas de panne.

COLLATE

Indique le classement par défaut de la base de données. Le nom de classement peut être un classement SQL ou Windows. S'il n'est pas précisé, c'est le classement par défaut de l'instance SQL Server qui est utilisé.

FOR ATTACH

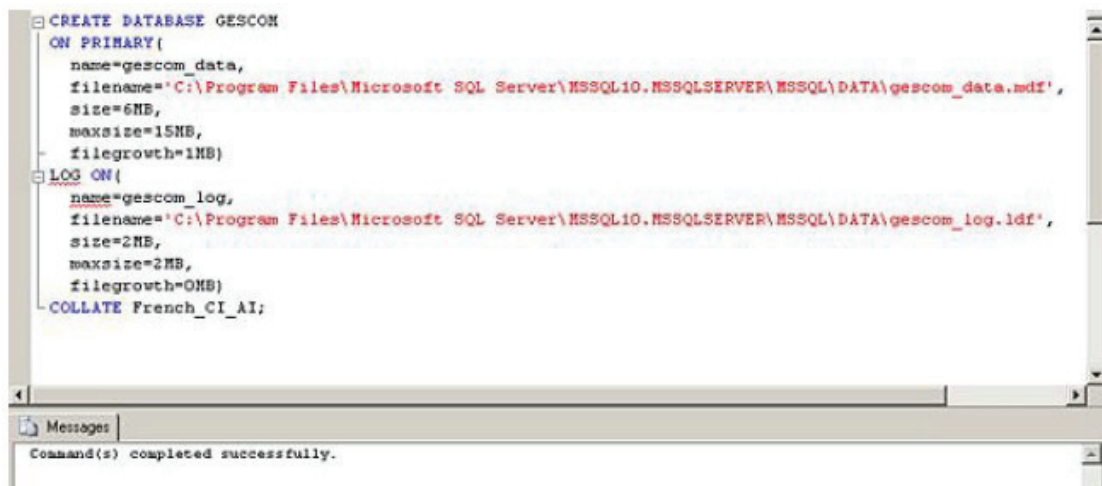
Pour créer une base en utilisant des fichiers déjà créés. Cette commande est utile lorsque la base est créée avec plus de 16 fichiers.

FOR ATTACH_REBUILD_LOG

Avec cette option, il est possible de créer la base en lui attachant les fichiers de données (mdf et ndf) mais pas nécessairement les fichiers journaux. Les fichiers journaux sont alors reconstruits à vide. Si une base est attachée de cette façon, il est important d'effectuer rapidement une sauvegarde complète de la base et de planifier tous les processus de sauvegarde. En effet, il n'est pas possible de s'appuyer sur les sauvegardes faites avant l'attachement car les séquences des journaux ne correspondent plus.

Exemple

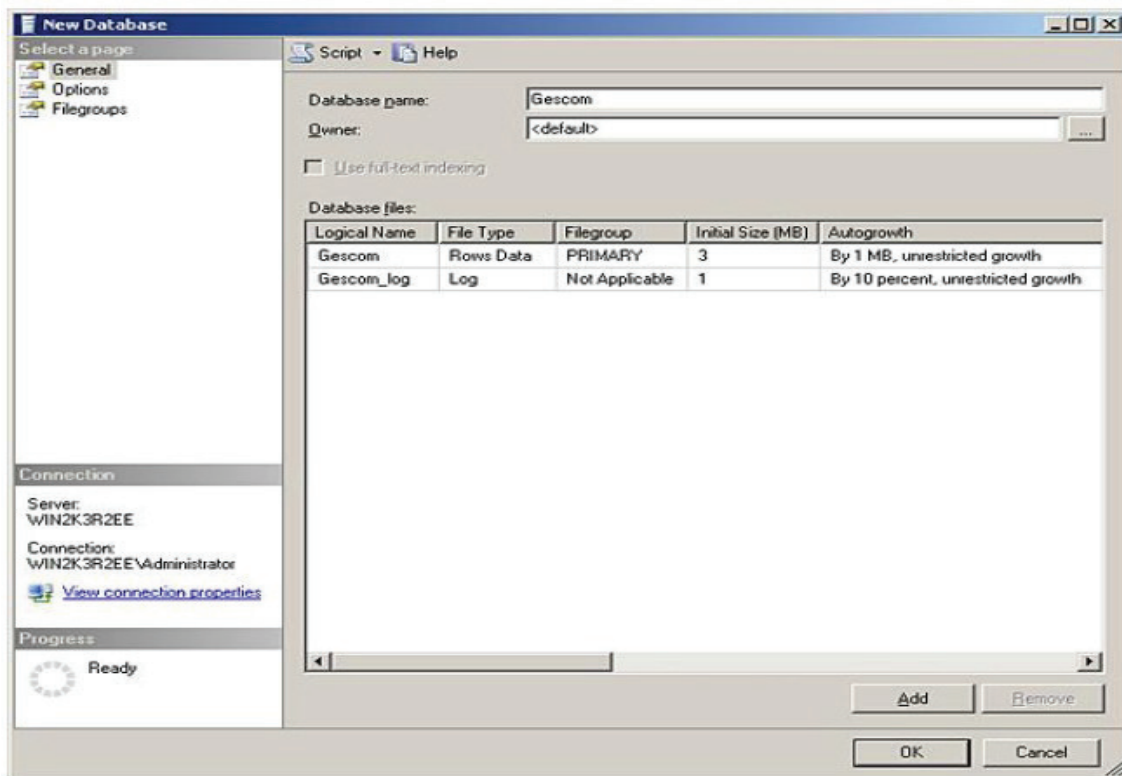
Création de la base de données Gescom (6 Mo) avec le journal de transaction (2 Mo).



```
CREATE DATABASE GESCOM
ON PRIMARY(
    name=gescom_data,
    filename='C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\gescom_data.mdf',
    size=6MB,
    maxsize=15MB,
    filegrowth=1MB)
LOG ON(
    name=gescom_log,
    filename='C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\gescom_log.ldf',
    size=2MB,
    maxsize=3MB,
    filegrowth=0MB)
COLLATE French_CI_AI;
```

Messages
Command(s) completed successfully.

Il est bien sûr possible de réaliser cette opération depuis la console graphique SQL Server Management Studio. Pour cela, après avoir sélectionné le noeud Bases de données depuis l'explorateur, il faut faire le choix **New Database** (Nouvelle Base de données) depuis le menu contextuel pour voir l'écran suivant s'afficher.



Depuis cette boîte de dialogue, il est possible de définir les différentes options de création de la base de données.

b. Modifier la taille

Il est possible d'augmenter ou de diminuer la taille des fichiers de façon automatique ou manuelle. Si un pas d'incrément (FILEGROWTH) et une taille maximum sont précisés lors de la création du fichier, le fichier changera de taille en fonction des besoins.

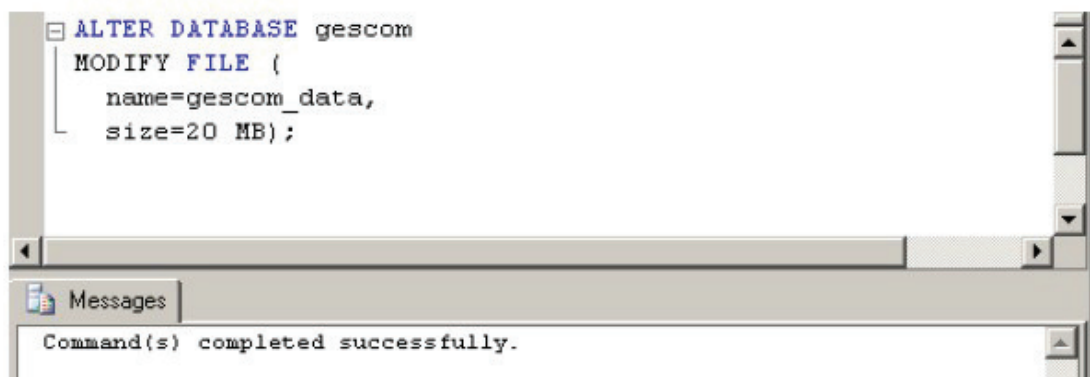
Il est possible de modifier manuellement la taille, la taille maximale et le taux d'augmentation d'un fichier de données avec la commande ALTER DATABASE.

Syntaxe

ALTER DATABASE nom
MODIFY FILE
(NAME=nom Logique
[,SIZE=taille]
[,MAXSIZE=taille Maxi]
[FILEGROWTH=valeur Incrément])

Exemple

Augmenter la taille d'un fichier existant :



Il est également possible d'ajouter des fichiers.

ALTER DATABASE nom
ADD FILE (
NAME = nom Logique,
FILENAME = 'nom Physique'
[, SIZE = taille]
[, MAXSIZE = { taille Maxi | UNLIMITED }]
[, FILEGROWTH = valeur rIncrément])

Exemple

Ajout d'un deuxième fichier à la base GESCOM :



```
ALTER DATABASE GESCOM
ADD FILE(
    name=gescom_data2,
    filename='C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\gescom_data2.ndf',
    size=3MB
);
```

Messages
Command(s) completed successfully.

La commande ALTER DATABASE permet une action beaucoup plus importante sur la base que la simple modification de taille des fichiers. Il est ainsi possible d'ajouter et de supprimer des fichiers et des groupes de fichiers, de modifier le nom de la base, de préciser le mode de fin par défaut des transactions en cours et de changer le classement de la base.

➤ Pour connaître les différents classements disponibles sur le serveur, il faut exécuter la requête suivante : `SELECT * FROM ::fn_helpcollations()`

c. Diminuer la taille

La taille des fichiers peut diminuer de façon automatique si l'option **autoshrink** a été positionnée sur la base.

Les commandes DBCC SHRINKFILE et DBCC SHRINKDATABASE permettent d'exécuter manuellement la diminution de taille.

DBCC SHRINKFILE ne va porter que sur un fichier en particulier tandis que DBCC SHRINKDATABASE va scruter tous les fichiers de la base.

L'opération de réduction des fichiers commence toujours par la fin du fichier. Par exemple, si la base dispose d'un fichier de 500 Mo que l'on souhaite ramener à une taille de 400 Mo ce sont les 100 derniers mégaoctets du fichier qui vont être réorganisés afin de ne plus détenir aucune donnée avant leur libération.

La quantité d'espace réellement libérée est fonction de la taille idéale fixée en paramètre à DBCC SHRINKFILE et de la réalité des données. Si dans l'exemple précédent, le fichier contient 450 Mo d'extensions utilisés, alors seulement 50 Mo d'espace seront libérés.

Syntaxe



```
DBCC SHRINKFILE (nom_fichier {[ ,taille_cible ]  
| [ , { EMPTYFILE | NOTRUNCATE | TRUNCATEONLY } ] ] )
```

```
DBCC SHRINKDATABASE (nom_base [ , pourcentage_cible ]  
[ , { NOTRUNCATE | TRUNCATEONLY } ] )
```

DBCC SHINKFILE

Réduit la taille du fichier de données ou du fichier journal pour la base de données spécifiée.

DBCC SHRINKDATABASE

Réduit la taille des fichiers de données dans la base de données spécifiée.

Taille_cible

Indique la taille souhaitée du fichier après réduction.

Pourcentage_cible

Indique le pourcentage d'espace libre que l'on souhaite obtenir dans le fichier de données après réduction de la base.

EMPTYFILE

Permet de demander à la commande DBCC_SHRINKFILE de transférer toutes les données présentes dans ce fichier de données vers un autre fichier du même groupe. Une fois vide, le fichier pourra être supprimé à l'aide d'une commande ALTER TABLE.

NOTRUNCATE

Réorganise le fichier en positionnant les pages occupées en haut de fichier, mais il n'est pas diminué.

TRUNCATEONLY

Coupe le fichier sans faire aucune réorganisation du fichier.

c. Supprimer la base

La commande DROP DATABASE permet de supprimer la base. Les fichiers physiques sont également supprimés.

Si certains fichiers sont détachés de la base avant sa suppression alors ils ne seront pas supprimés et il sera nécessaire d'effectuer cette opération de façon manuelle depuis l'explorateur de fichiers. Enfin, si des utilisateurs sont connectés sur la base il n'est pas possible de la supprimer. Pour forcer la déconnexion de ces derniers et permettre la suppression de la base il est nécessaire de basculer vers le mode SINGLE_USER à l'aide de l'instruction ALTER DATABASE.

L'instruction DROP DATABASE ne peut être exécutée que si le mode autocommit est activé (ce qui est le cas par défaut). Il n'est pas possible de supprimer les bases système.

d. Renommer une base

Il est possible de renommer une base de données par l'intermédiaire de l'instruction ALTER DATABASE.

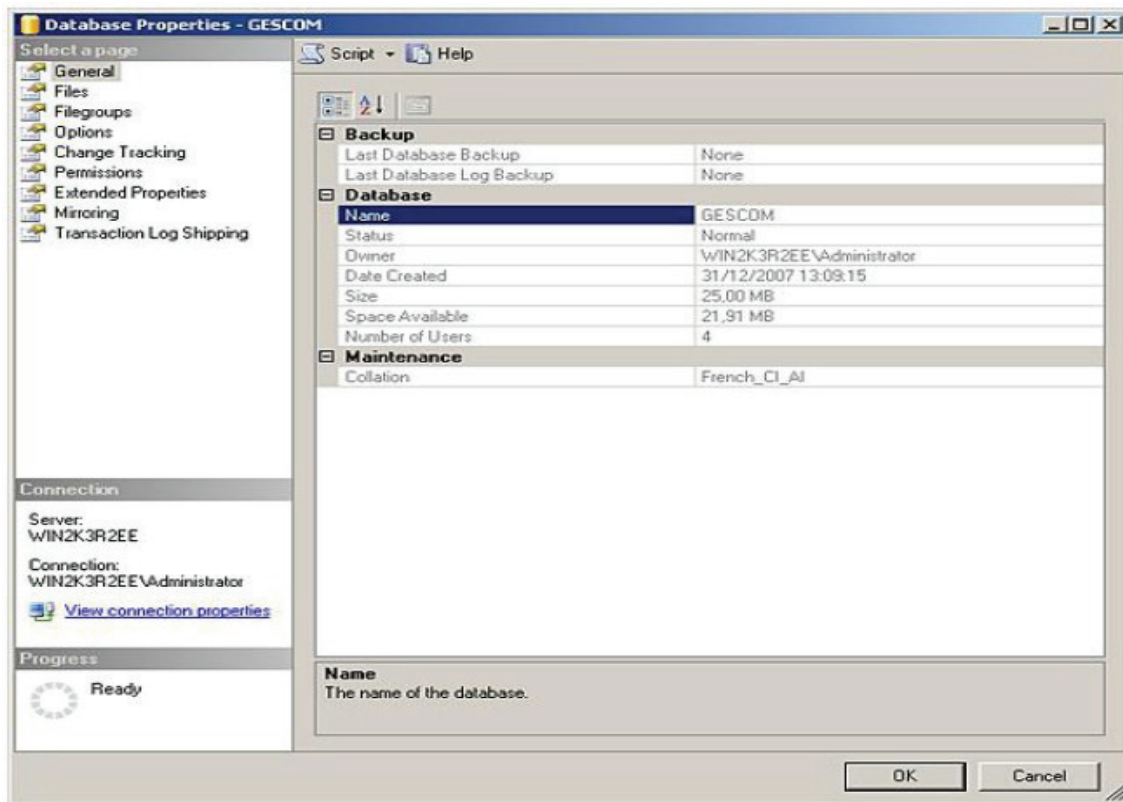
Syntaxe

ALTER DATABASE *nom Base* MODIFY NAME= *nouveau NomBase*

➤ La procédure `sp_renamedb` est maintenue pour des raisons de compatibilité ascendante. Il faut veiller à ne plus l'utiliser dans SQL Server 2008.

e. Configurer une base

Il est possible de configurer une base de données pour fixer un certain nombre d'options afin d'obtenir le comportement souhaité de la base en fonction des besoins des utilisateurs. On accède à ces différentes options, soit de façon graphique par SQL Server Management Studio en se positionnant sur la base puis en appelant la fenêtre des propriétés par la touche [F4], soit par le menu contextuel associé à la base, ou bien encore par le menu **View Properties Window** (Affichage Fenêtre Propriétés) dans le menu général de SQL Server Management Studio.



Syntaxe

ALTER DATABASE nomBase SET option ;

Etat de la base

ONLINE

Permet de rendre la base de données de nouveau visible.

OFFLINE

Permet de rendre inaccessible la base de données. La base de données est fermée et arrêtée proprement. Il n'est pas possible de faire des opérations de maintenance sur une base de données offline.

EMERGENCY

La base de données est mise en lecture seule, la journalisation est désactivée et son accès est limité aux seuls administrateurs du serveur.

Accès

SINGLE_USER

Accès limité à un seul utilisateur.

RESTRICTED_USER

Seuls les membres des rôles db_owner, dbcreator, ou sysadmin peuvent se connecter à la base.

MULTI_USER

C'est le mode par défaut qui permet à tous les utilisateurs disposant de privilèges suffisants d'accéder à l'information.

Opérations possibles

READ_ONLY

La base est accessible uniquement pour les opérations de lecture.

READ_WRITE

La base est accessible pour les opérations de lecture/écriture.

Accès

DBO use only

La base n'est accessible que par son propriétaire.

Paramétrage

ANSI_NULL_DEFAULT

Définit la valeur par défaut de la contrainte de nullité de colonne. Selon la norme ANSI, une colonne peut être NULL par défaut.

RECURSIVE_TRIGGERS

Autorise la récursivité des triggers.

TORN_PAGE_DETECTION

Permet de détecter les pages incomplètes.

AUTO_CLOSE

La base est arrêtée et les ressources sont libérées après la déconnexion du dernier utilisateur.

AUTO_SHRINK

Les fichiers de la base pourront être automatiquement réduits.



AUTO_CREATE_STATISTICS

Toutes les statistiques manquantes lors de l'optimisation d'une requête sont créées. Cette option est active (positionnée à ON) par défaut.

AUTO_UPDATE_STATISTICS

Toutes les statistiques obsolètes pour la bonne optimisation d'une requête sont recalculées.

AUTO_UPDATE_STATISTICS_ASYNC

Les statistiques qui permettent de représenter la pertinence des index sont mises à jour de façon asynchrone. La requête qui provoque la mise à jour des statistiques n'attend pas que les statistiques soient à jour pour s'exécuter.

Ce sont les futures requêtes qui profiteront de cette mise à jour. Cette option est active (positionnée à ON) par défaut.

QUOTED_IDENTIFIERS

Les identificateurs délimités peuvent être encadrés par des guillemets doubles.

ANSI_NULLS

Si le paramètre est vrai (true), alors toutes les comparaisons avec une valeur NULL sont évaluées à inconnues. Si le paramètre est faux (false) alors les comparaisons avec les valeurs NULL et les valeurs non unicode sont évaluées à VRAI si les deux valeurs sont NULL.

ANSI_WARNINGS

Permet de faire remonter des messages d'erreur ou des avertissements lorsque certaines conditions sont remplies.

ARITHABORT

Permet d'arrêter le traitement du lot d'instructions lors d'un dépassement de capacité ou d'une division par zéro.

CONCAT_NULL_YIELDS_NULL

Le résultat est NULL si l'un des deux opérandes d'une opération de concaténation est NULL.

CURSOR_CLOSE_ON_COMMIT

Permet de fermer tous les curseurs lors de la définition d'une transaction ou lors de la fin d'une transaction.

CURSOR_DEFAULT

Les déclarations de curseur ont pour valeur par défaut LOCAL.

NUMERIC_ROUNDABORT

Une erreur est levée si une perte de précision intervient au cours d'un calcul.

RECOVERY



Permet de préciser la stratégie de sauvegarde planifiée au niveau de la base. Ce paramètre a une incidence directe sur les informations conservées dans les journaux de transactions.

PAGE_VERIFY

Cette option permet de valider la qualité des informations stockées au niveau de chaque page. L'option par défaut CHECKSUM est celle recommandée par SQL Server.

SUPPLEMENTAL_LOGGING

En positionnant cette option à ON (OFF par défaut), des informations complémentaires vont être ajoutées au journal. Il est possible de connaître l'état de cette option en examinant la valeur contenue dans la colonne `is_supplemental_logging_enabled` de la vue `sys.databases`.

PARAMETERIZATION

En mode SIMPLE, par défaut les requêtes sont paramétrées en fonction des règles en vigueur sur le serveur. En utilisant le mode FORCED, SQL Server paramètre toutes les requêtes avant de dresser le plan d'exécution.

Gestion des transactions

ROLLBACK AFTER nombre

L'annulation des transactions est effective après nombre secondes d'attente.

ROLLBACK IMMEDIATE

L'annulation de la transaction est immédiate.

NO_WAIT

Si la transaction n'accède pas immédiatement aux ressources qui lui sont nécessaires, elle est annulée.

ANSI_PADDING

Permet de spécifier si les espaces à droite sur les données de type caractère doivent être supprimés ou pas.

COMPATIBILITY

Permet de fixer le niveau de compatibilité de la base de données : 80 pour SQL Server 2000, 90 pour SQL Server 2005 et 100 pour SQL Server 2008.

DATE_CORRELATION_OPTIMISATION

Avec cette option SQL Server se charge de maintenir la corrélation des statistiques entre deux tables liées par une contrainte de clé étrangère et qui possèdent toutes les deux une colonne de type datetime.



Accès externe

DB_CHAINING

Permet de gérer les contextes de sécurité lors de l'accès à la base à partir d'une autre base.

TRUSTWORTHY

Des modules internes (procédures stockées, fonctions) peuvent accéder à des ressources externes au serveur en utilisant un contexte impersonnate.

Service Broker

ENABLE_BROKER

Active le service Broker.

DISABLE_BROKER

Désactive le service Broker.

NEW_BROKER

Permet de préciser que la base doit recevoir un nouvel identifiant Service Broker.

ERROR_BROKER_CONVERSATIONS

Les conversations en cours vont recevoir un message d'erreur et vont ainsi être clôturées.

Snapshot (Capture instantanée)

ALLOW_SNAPSHOT_ISOLATION

Lorsque ce mode est activé, toutes les transactions sont capables de travailler avec une capture instantanée (snapshot) de la base telle qu'elle était au début de la transaction.

READ_COMMITTED_SNAPSHOT

Lorsque ce mode est activé, toutes les instructions perçoivent les données telles qu'elles étaient au début de l'instruction.

Gérer les tables et les index

1. Identifiant

Tous les éléments créés dans SQL Server sont parfaitement identifiés par leur nom qui est utilisé en tant qu'identifiant.

En effet, deux objets de même type ne peuvent pas avoir le même nom s'ils sont définis au même niveau. Par exemple, sur une instance de SQL Server, il n'est pas possible d'avoir deux



bases de données avec le même nom, par contre, c'est totalement possible si les bases sont définies sur deux instances distinctes de SQL Server.

De même, au sein d'une base de données, il n'est pas possible d'avoir deux tables avec le même nom. C'est à partir de l'identifiant qu'il est possible de manipuler via le SQL les objets. Il est donc important de définir correctement ces identifiants.

Les identifiants sont composés de 1 à 128 caractères. Ils commencent toujours par une lettre ou l'un des caractères suivants : `_`, `@`, `#`.

Les caractères suivants sont des caractères alphanumériques.

Bien entendu aucun identifiant ne peut correspondre à un mot clé du Transact SQL.

Il existe deux catégories d'identifiants : les réguliers et les délimités.

Les identifiants réguliers

Cette catégorie d'identifiant est la plus communément utilisée et c'est celle à privilégier. En effet, ce type d'identifiant est présent dans toutes les bases et présente beaucoup de souplesse au niveau de l'écriture des requêtes car la casse n'est pas conservée.

Exemple :

RessourcesInformatiques

Les identifiants délimités

Cette catégorie d'identifiant permet de conserver des caractères spéciaux dans les identifiants comme les caractères accentués, les espaces, ... mais également de conserver la casse. Ces identifiants sont utilisés entre des crochets `[]` ou bien des guillemets `""`. L'utilisation de ce type d'identifiant ne permet que rarement de gagner en clarté car l'écriture des requêtes est plus lourde. Il est donc préférable d'utiliser des identifiants réguliers.

Exemple :

[éditions numéro 2], "Ressources Informatiques", ...

2. Les types de données

Lors de la définition d'une colonne, on précisera le format d'utilisation de la donnée ainsi que le mode de stockage par le type de la colonne.

a. Types de données système



Ces types sont disponibles pour toutes les bases de données en standard.

Caractères

char[(n)]

Chaîne de caractères de longueur fixe, de n caractères maximum. Par défaut 1, maximum 8000.

varchar(n|max)

Chaîne de caractères à longueur variable, de n caractères maximum. Par défaut 1, maximum 8000 caractères. En précisant max, la variable peut contenir des données de type texte allant jusqu'à 2^{31} caractères.

nchar[(n)]

Chaîne de caractères unicode, maximum 4000 caractères.

nvarchar (n|max)

Chaîne de caractères unicode, maximum 4000. En précisant max, la variable peut contenir des données de type texte allant jusqu'à 2^{31} octets.

Numériques

decimal [(p,d)]

Numérique exact de précision p (nombre de chiffres total), avec d chiffres à droite de la virgule.

p est compris entre 1 et 38, 18 par défaut.

d est compris entre 1 et p, 0 par défaut.

Exemple : pour décimal (8,3) l'intervalle admis sera de 99999,999
à +99999,999.

Les valeurs sont gérées de 1038

à 1038 1.

numeric [(p,d)]

Identique à decimal. Pour le type decimal, la précision pourra être parfois plus grande que celle requise.

bigint

Type de données entier codé sur 8 octets. Les valeurs stockées avec ce type de données sont comprises entre (-9223 372 036 854 775 808) et (9 223 372 036 854 775 807).

int

Nombre entier entre (-2147783648)et (+2147483647). Le type de données **int** est spécifique SQL Server et son synonyme **integer** est quant à lui compatible ISO.

smallint

Nombre entier entre (32768)et (+32767).

tinyint

Nombre entier positif entre 0 et 255.

float[(n)]

Numérique approché de n chiffres, n allant de 1 à 53.

real



Identique à float(24).

money

Numérique au format monétaire compris entre
337 203 685 477, 5808 et + 337 203 685 477, 5807 (8 octets).

smallmoney

Numérique au format monétaire compris entre
748,3648 et + 748,3647 (4 octets).

Binaires

binary[(n)]

Donnée binaire sur n octets (1 à 255), la longueur est fixe.

varbinary (n|max)

Donnée binaire de longueur variable de n octets (1 à 8000). L'option max permet de réserver un espace de 231 octets au maximum.

Date

Pour la gestion des données de types date et heure SQL Server 2008 propose de nouveaux types de données afin d'optimiser le stockage des données. Ces nouveaux types de données sont introduits pour permettre une gestion plus fine des données de types date et heure.

Tout d'abord, il existe des types particuliers pour stocker les données de type heure et d'autres types pour stocker les données de type date. Cette séparation est bénéfique car elle permet de donner plus de précision tout en limitant l'espace utilisé par les données de tel ou tel type.

Par exemple, Est-il nécessaire de conserver des données de type heures, minutes et secondes lorsque seule la date de naissance d'un client doit être conservée ? Sans aucun doute, non. Le simple fait de conserver ces informations peut induire des erreurs lorsque par la suite des calculs vont être faits. Il est donc plus raisonnable et plus performant d'adopter pour cette donnée, un type qui ne conserve que la date.

Ces nouveaux types de données, pour les données de type date et heure, vont également permettre une meilleure compatibilité avec les autres systèmes de gestion de données et faciliter les opérations de reprise de données.

Bien entendu SQL Server offre la possibilité avec les types datetime2 et datetimeoffset de conserver des informations de type date et heure de façon simultanée.

Partie 1

Le type `datetime offset` permet non seulement de stocker des informations de type date et heure avec une précision pouvant aller jusqu'à 100 nanosecondes, mais en plus c'est l'heure au format UTC qui est conservée ainsi que le décalage (en nombre d'heures) entre cette heure UTC et la zone horaire depuis laquelle travaille l'utilisateur qui introduit les informations dans la base.

➤ Les types `datetime` et `smalldatetime` sont toujours présents dans SQL Server, mais il est préférable de privilégier les types `time`, `date`, `datetime2` et `datetimeoffset` dans les nouveaux développements. En effet, ces types offrent plus de précision et un meilleur respect des standards SQL.

datetime

Permet de stocker une date et une heure sur 8 octets. 4 pour un nombre de jours par rapport au 1er janvier 1900, 4 pour un nombre de millisecondes après minuit. Les dates sont gérées du 1er janvier 1753 au 31 décembre 9999. Les heures sont gérées avec une précision de 3,33 millisecondes.

smalldatetime

Permet de stocker une date et une heure sur 4 octets. Les dates sont gérées du 1er janvier 1900 au 6 juin 2079, à la minute près.

datetime2

Plus précis que le type `datetime`, il permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes.

datetimeoffset

Permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes. Les informations horaires sont stockées au format UTC et le décalage horaire est conservé afin de retrouver l'heure locale renseignée initialement.

date

Permet de stocker une date comprise entre le 01/01/0001 et le 31/12/9999 avec une précision d'une journée.

time

Permet de stocker une donnée positive de type heure inférieure à 24h00 avec une précision de 100 nanosecondes.

Spéciaux

bit

Valeur entière pouvant prendre les valeurs 0, 1 ou null.

timestamp

Donnée dont la valeur est mise à jour automatiquement lorsque la ligne est modifiée ou insérée.

uniqueidentifier

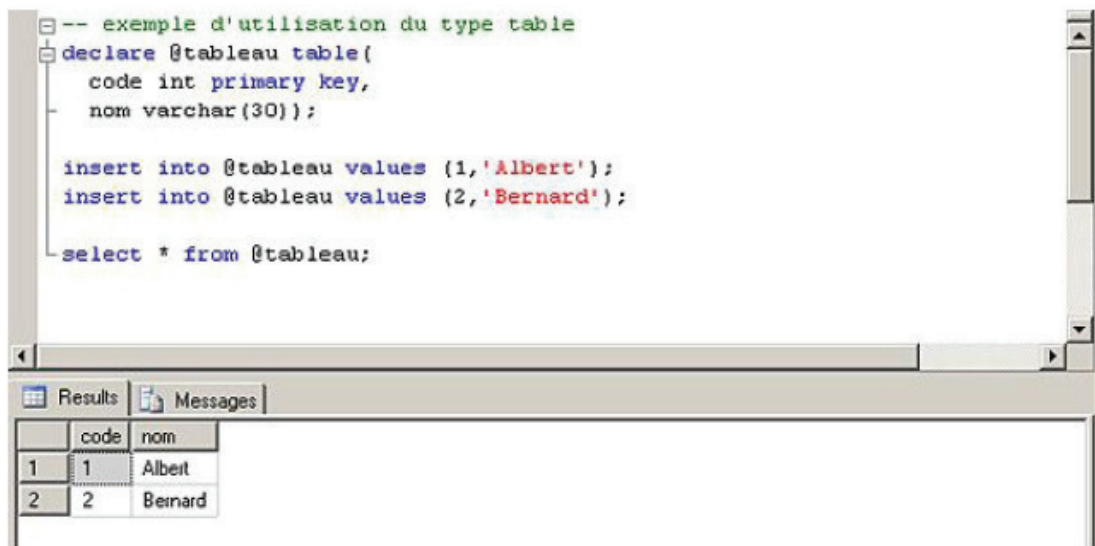
Permet de créer un identificateur unique en s'appuyant sur la fonction NEWID().

sql_variant

Le type de données **sql_variant** permet de stocker n'importe quel type de données à l'exception des données de type **text**, **ntext**, **timestamp** et **sql_variant**. Si une colonne utilise ce type de données, les différentes lignes de la table peuvent stocker dans cette colonne des données de type différent. Une colonne de type **sql_variant** peut posséder une longueur maximale de 8016 octets. Avant d'utiliser une valeur stockée au format **sql_variant** dans une opération, il est nécessaire de convertir les données dans leur format d'origine. Les colonnes utilisant le type **sql_variant** peuvent participer à des contraintes de clés primaires, de clés étrangères ou d'unicité, mais les données contenues dans la clé d'une ligne ne peuvent excéder les 900 octets (limite imposée par les index). **sql_variant** ne peut pas être utilisé dans les fonctions CONTAINSTABLE et FREETEXTTABLE.

table

C'est un type de données particulier qui permet de stocker et de renvoyer un ensemble de valeurs en vue d'une utilisation future. Le mode principal d'utilisation de ce type de données est la création d'une table temporaire.



```
-- exemple d'utilisation du type table
declare @tableau table(
    code int primary key,
    nom varchar(30));

insert into @tableau values (1,'Albert');
insert into @tableau values (2,'Bernard');

select * from @tableau;
```

	code	nom
1	1	Albert
2	2	Bernard

xml

Ce type permet de stocker un document xml dans une colonne au sein d'une table relationnelle.

➤ Les types **text**, **ntext** et **images** sont maintenus pour des raisons de compatibilité. Il faut maintenant leur préférer **varchar(max)** et **varbinary(max)**.

b. Types de données définis par l'utilisateur

Il est possible de définir ses propres types de données, soit par l'intermédiaire de Management Studio, soit par la commande CREATE TYPE.

Les procédures stockées `sp_addtype` et `sp_droptype` sont maintenues pour des raisons de compatibilité ascendante. Microsoft recommande de ne plus les utiliser car elles ne seront sans doute plus présentes dans les versions à venir de SQL Server.

Syntaxe (création)

```
CREATE TYPE nomType  
{FROM typeDeBase [ ( longueur [ , precision ] ) ]  
[ NULL | NOT NULL ]  
| EXTERNAL NAME nomAssembly [ .nomClasse ]  
} [ ; ]
```

Il est possible de définir un type de données en s'appuyant sur la définition d'une classe. Cette option est liée à l'intégration du CLR dans SQL Server. Cette intégration est détaillée plus tard dans ce syllabus.

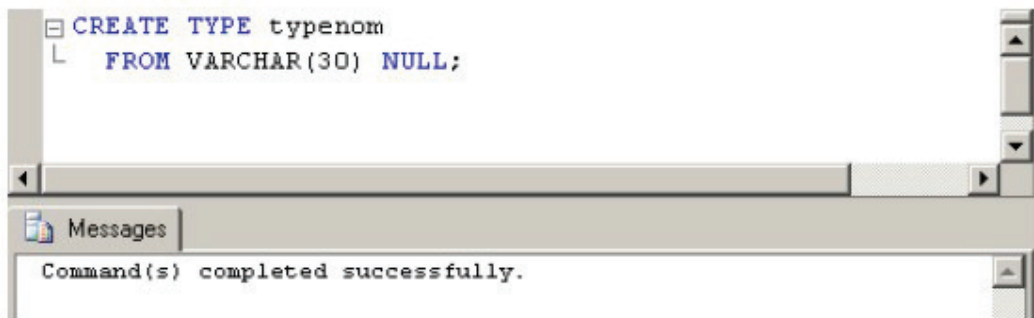
Syntaxe (suppression)

```
DROP TYPE [ schema_name. ] type_name [ ; ]
```

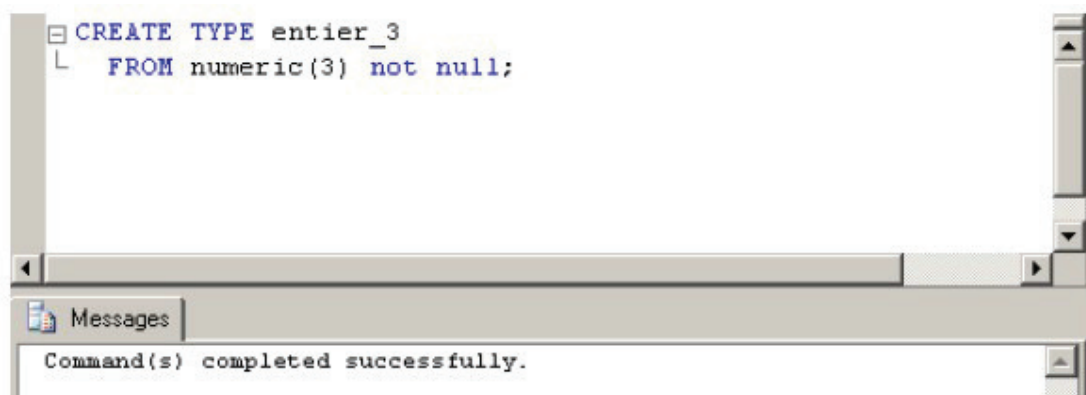
Un type ne pourra pas être supprimé s'il est utilisé dans une table de la base où il a été créé.

Exemples

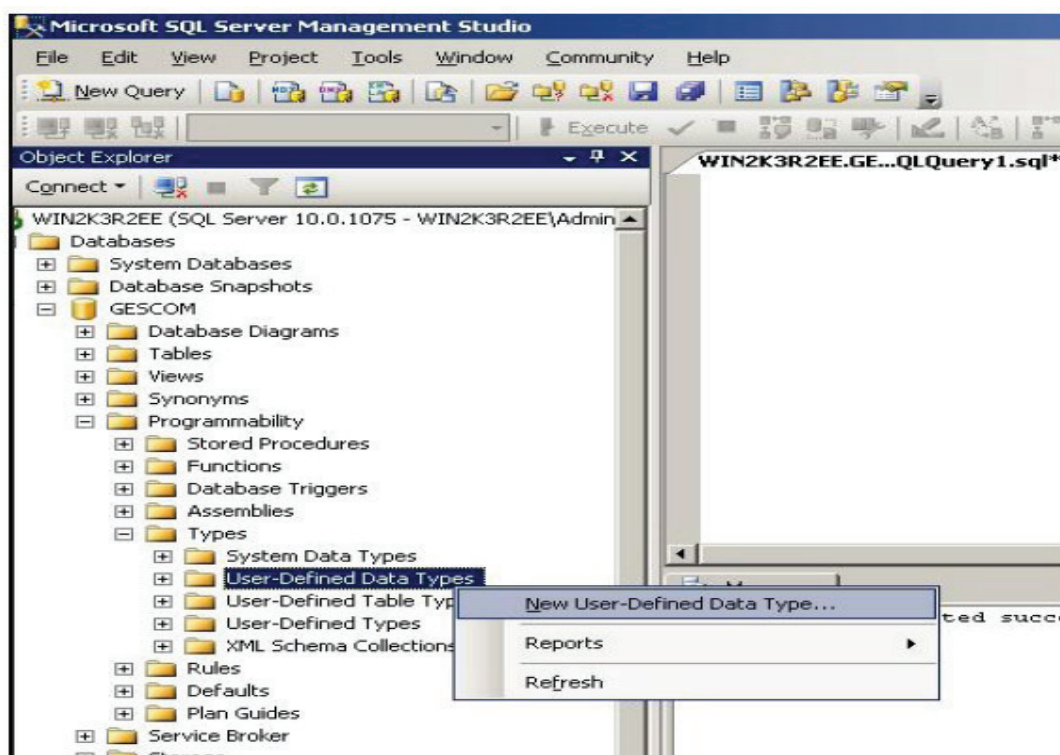
Création d'un type pour les colonnes telles que nom client, nom fournisseur etc. :



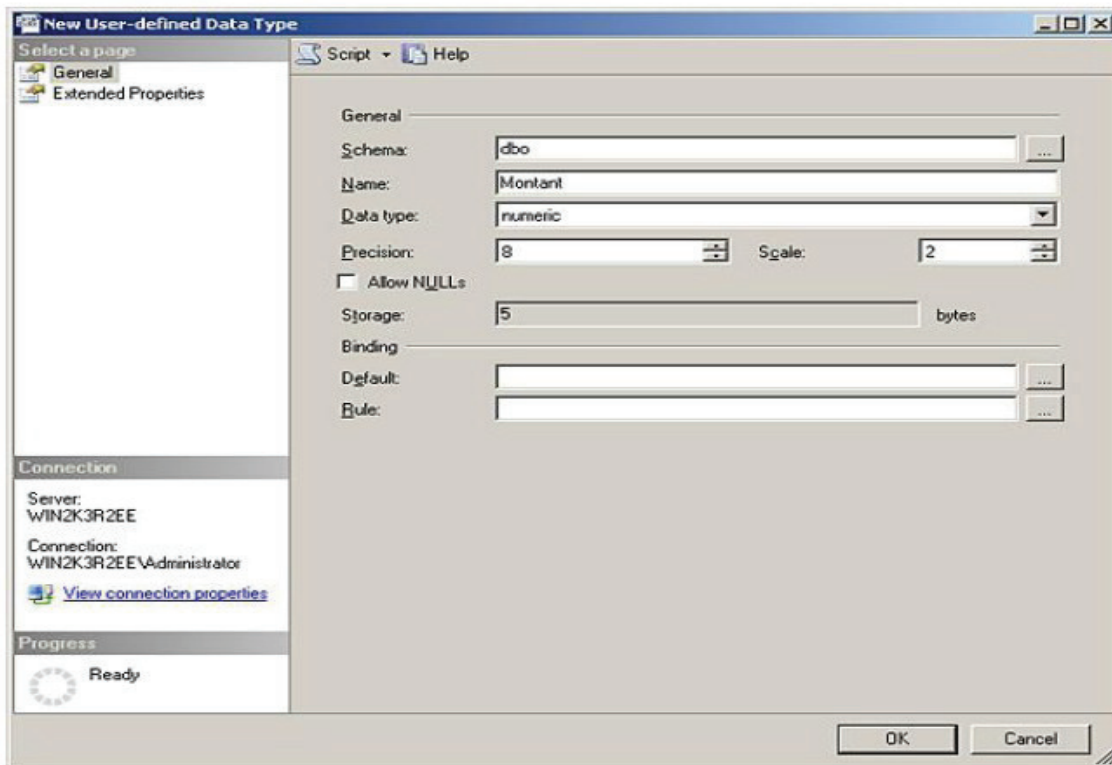
*Création d'un type pour des valeurs entre 999
et +999 :*



Demander la création d'un nouveau type de données depuis SQL Server Management Studio :



Définition du nouveau type de données "Montant" depuis SQL Server Management Studio :



En plus de définir des alias (comme illustré cidessus), l'instruction CREATE TYPE permet de créer des types UDT (*User Defined Type*) c'est à dire définis à l'aide du CLR (ce point sera abordé dans le chapitre CLR).

Au niveau Transact SQL, l'instruction CREATE TYPE permet également de créer des types composés de plusieurs champs. Ces types sont fréquemment nommés types structurés dans les langages de programmation.

En ce qui concerne SQL Server, l'instruction CREATE TYPE permet de créer un type TABLE ou tableau. Chaque colonne qui participe à la définition de ce nouveau type est définie sur le même principe qu'une colonne dans une table. Ainsi il est possible de définir les contraintes d'intégrité de clé primaire (PRIMARY KEY), d'unicité (UNIQUE), de validation (CHECK) et de non nullité. Ces contraintes peuvent être définies au niveau de la colonne ou bien de la table. Il est

également possible de définir une colonne de type identité.

L'instruction CREATE TYPE permet ainsi de créer des types dits fortement typés car les contraintes d'intégrité permettent une définition plus précise du format possible des données. L'introduction de ce nouveau type va permettre de définir des paramètres de fonctions ou procédures de type tableau. On parlera alors d'un table value parameter.

Syntaxe

```
CREATE TYPE nom Type AS TABLE (  
colonne type Colonne[contrainte Colonne], ...)
```

nomType

Nom du type ainsi créé.

colonne

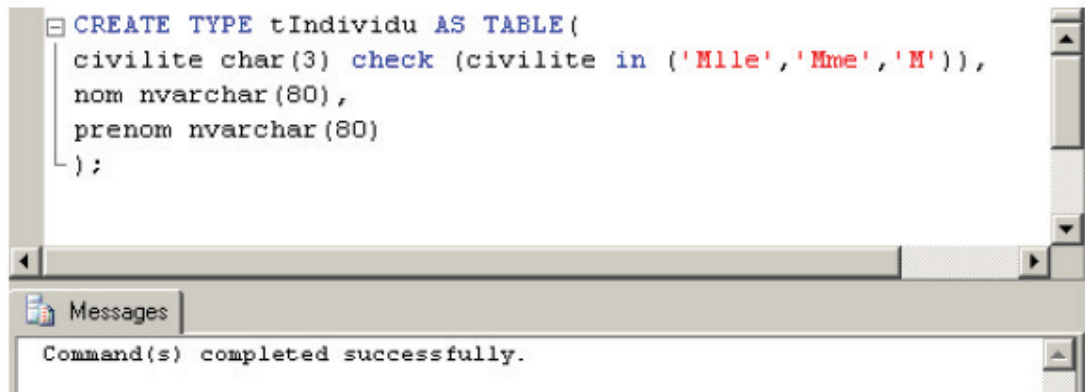
Nom de la colonne qui participe à la définition de ce nouveau type. Il est bien sûr possible de définir plusieurs colonnes.

typeColonne

Type de données Transact SQL sur lequel est définie la colonne. Toutes les colonnes d'un même type de table ne sont pas forcément définies sur le même type ni avec la même précision. Définition de la contrainte d'intégrité associée à la colonne.

Exemple

Dans l'exemple suivant un type représentant un individu est défini. Ce type est composé des champs civilité, nom et prénom. Pour le champ civilité seules certaines valeurs sont autorisées.



```
CREATE TYPE tIndividu AS TABLE(  
    civilite char(3) check (civilite in ('Mlle', 'Mme', 'M')),  
    nom nvarchar(80),  
    prenom nvarchar(80)  
);
```

Messages
Command(s) completed successfully.

3. Gérer les tables

Une table représente une structure logique dans laquelle les données vont être rangées. Pour permettre une bonne organisation des informations, chaque table est constituée de colonnes afin de structurer les données. Chaque colonne est parfaitement identifiée par son nom, qui est unique à l'intérieur de la table, et par son type de données. Les données sont réparties entre plusieurs tables. Les contraintes d'intégrité permettent de garantir la cohérence des données. Les trois opérations de gestion de table sont la création (CREATE TABLE), la modification (ALTER TABLE) et la suppression (DROP TABLE). Ces opérations peuvent être réalisées en Transact SQL ou par SQL Server Management Studio par un utilisateur "dbo" ou ayant reçu le droit CREATE TABLE.

a. Créer une table

L'étape de création des tables est une étape importante de la conception de la base car les données sont organisées par rapport aux tables. Cette opération est ponctuelle et elle est en général réalisée par l'administrateur (DBA : *DataBase Administrator*) ou tout au moins par la personne chargée de la gestion de la base. La création d'une table permet de définir les colonnes (nom et type de données) qui la composent ainsi que les contraintes d'intégrité. De plus, il est possible de définir des colonnes calculées, un ordre de tri spécifique à la colonne ainsi que la destination des données de type texte ou image.

Syntaxe

```
CREATE TABLE [nom Schema.] nom_table  
( nom_colonne {type colonne|AS expression_calculée}  
[,nom_colonne ... ][,contraintes...])  
[ON groupefichier]  
[TEXTIMAGE_ON groupe_fichier]
```

nomSchema

Nom du schéma dans lequel la table va être définie.

nom_table

Peut être sous la forme base.propriétaire.table.

nom_colonne

Nom de la colonne qui doit être unique dans la table.

Typecolonne

Type système ou type défini par l'utilisateur.

contraintes

Règles d'intégrité (traitées ultérieurement dans ce syllabus).

groupefichier

Groupe de fichiers sur lequel va être créée la table.

AS expression_calculée

Il est possible de définir une règle de calcul pour les colonnes qui contiennent des données calculées. Bien entendu, ces colonnes ne sont accessibles qu'en lecture seule, et il n'est pas possible d'insérer des données ou de mettre à jour les données d'une telle colonne.

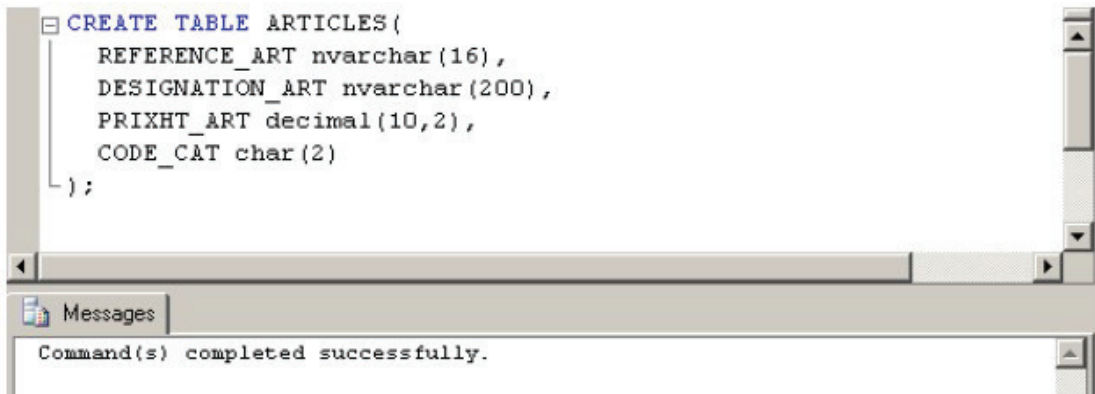
TEXTIMAGE_ON

Permet de préciser le groupe de fichiers destination pour les données de type texte et image. Il est possible de créer 2 milliards de tables par base de données.

➤ Le nombre maximal de colonnes par table est de 1024. La longueur maximale d'une ligne est de 8060 octets (sans compter les données texte ou image).

Exemples

Création de la table ARTICLES :



Affichage des informations sur la table à l'aide de la procédure stockée sp_help :

exec sp_help articles

	Name	Owner	Type	Created_datetime
1	ARTICLES	dbo	user table	2007-12-31 14:13:19.897

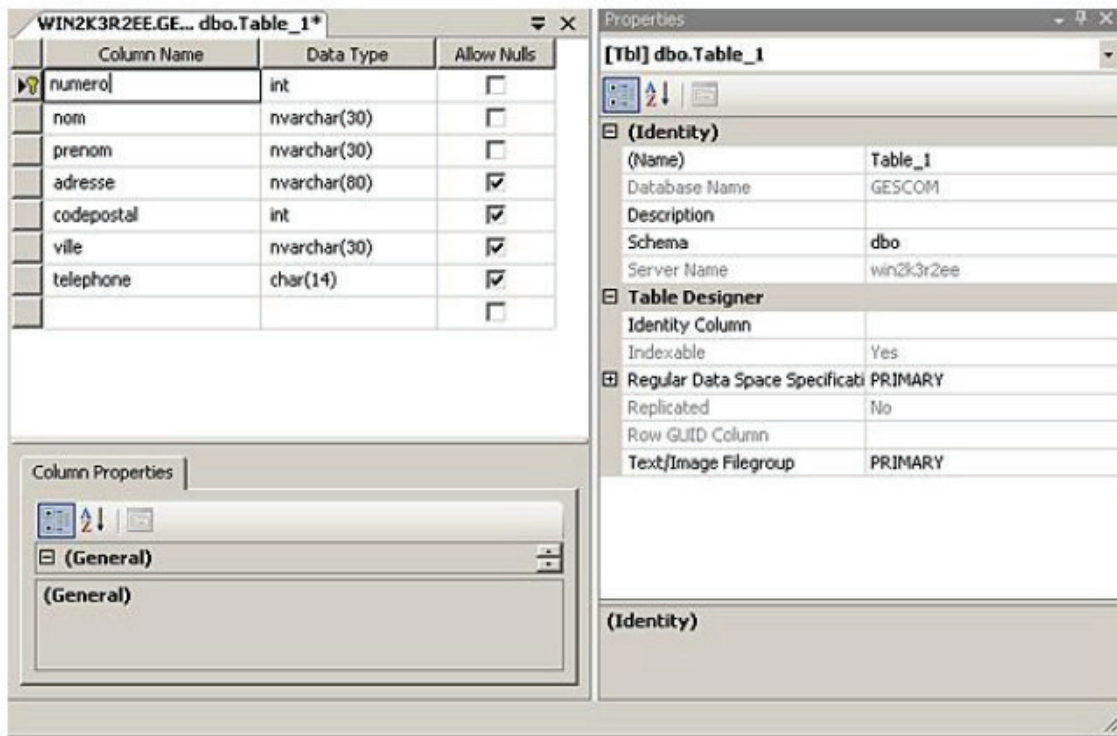
	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	F
1	REFERENCE_ART	char	no	4			yes	no	y
2	DESIGNATION_ART	varchar	no	20			yes	no	y
3	PRIXHT_ART	decimal	no	9	10	2	yes	[n/a]	y
4	CODE_CAT	char	no	2			yes	no	y

	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

Création de la table CLIENTS (depuis l'interface graphique) :



b. Modifier une table

La modification de table est effectuée par la commande ALTER TABLE ou par l'interface graphique de SQL Server Management Studio. Lors d'une modification de table, il est possible d'ajouter et de supprimer des colonnes et des contraintes, de modifier la définition d'une colonne (type de données, classement et comportement vis à vis de la valeur NULL), d'activer ou de désactiver les contraintes d'intégrité et les déclencheurs. Ce dernier point peut s'avérer utile lors d'import massif de données dans la base si l'on souhaite conserver des temps de traitements cohérents.

Syntaxe

```
ALTER TABLE [nomSchema.] nomtable
{ [ ALTER COLUMN nom_colonne
{ nouveau_type_données [ ( longueur [ , precision ] ) ]
[ COLLATE classement ] [ NULL | NOT NULL ] } ]
| ADD nouvelle_colonne
```

```
| [ WITH CHECK | WITH NOCHECK ] ADD contrainte_table  
| DROP { [ CONSTRAINT ] nom_contrainte | COLUMN nom_colonne }  
| { CHECK | NOCHECK } CONSTRAINT { ALL | nom_contrainte }  
| { ENABLE | DISABLE } TRIGGER { ALL | nom_déclencheur } }
```

nomSchema

Nom du schéma dans lequel la table va être définie.

WITH NOCHECK

Permet de poser une contrainte d'intégrité sur la table sans que cette contrainte soit vérifiée par les lignes déjà présentes dans la table.

COLLATE

Permet de définir un classement pour la colonne qui est différent de celui de la base de données.

NULL, NOT NULL

Permettent de définir une contrainte de nullité ou de non nullité sur une colonne existante de la table.

CHECK, NOCHECK

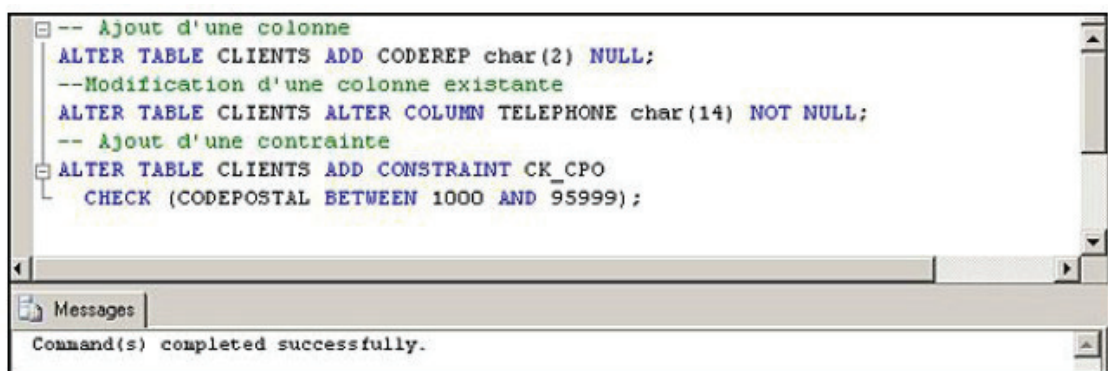
Permettent d'activer et de désactiver des contraintes d'intégrité.

ENABLE, DISABLE

Permettent d'activer et de désactiver l'exécution des déclencheurs associés à la table.

Exemple

Ajout d'une colonne :



```
-- Ajout d'une colonne  
ALTER TABLE CLIENTS ADD CODEREPE char(2) NULL;  
--Modification d'une colonne existante  
ALTER TABLE CLIENTS ALTER COLUMN TELEPHONE char(14) NOT NULL;  
-- Ajout d'une contrainte  
ALTER TABLE CLIENTS ADD CONSTRAINT CK_CPO  
CHECK (CODEPOSTAL BETWEEN 1000 AND 95999);
```

Messages
Command(s) completed successfully.

c. Supprimer une table

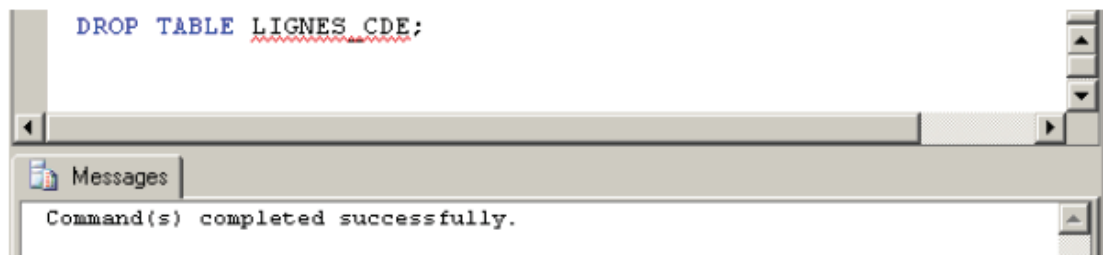
La suppression d'une table entraîne la suppression de toutes les données présentes dans la table. Les déclencheurs et les index associés à la table sont également supprimés. Il en est de même pour les permissions d'utilisation de la table. Par contre, les vues, procédures et fonctions qui référencent la table ne sont pas affectées par la suppression de la table. Si elles référencent la table supprimée, alors une erreur sera levée lors de la prochaine exécution.

Syntaxe

DROP TABLE [nomSchema.] nomtable [,nomtable...]

La suppression d'une table supprimera les données et les index associés. La suppression ne sera pas possible si la table est référencée par une clé étrangère.

Suppression d'une table :



d. Nom complet d'une table

En fonction de l'emplacement depuis lequel la table, et plus généralement l'objet, est référencé, il est nécessaire d'utiliser un nom plus ou moins précis. Le nom complet d'une table, et donc d'un objet, est de la forme suivante :

nomBase.nomSchema.nomObjet

Cependant, comme habituellement les objets référencés sont présents sur la base courante, il est possible d'omettre le nom de la base.

Le nom du schéma peut également ne pas être spécifié. Dans un tel cas de figure, le moteur de base de données cherchera l'objet dans le schéma associé à l'utilisateur, puis si cette recherche est infructueuse dans le schéma dbo.

Mise en oeuvre de l'intégrité des données

Pour assurer la cohérence des données dans la base, il est possible de gérer au niveau du serveur un ensemble de fonctionnalités qui permettent de centraliser les contrôles et les règles de fonctionnement dictés par l'analyse.

La mise en oeuvre de l'intégrité des données peut se faire de manière procédurale par les valeurs par défaut (DEFAULT) et les déclencheurs (TRIGGER) ou de manière déclarative par les contraintes (CONSTRAINT) et la propriété IDENTITY.

L'intégrité des données traduit les règles du modèle relationnel, règle de cohérence (intégrité de domaine), existence de valeurs nulles, règle d'unicité (intégrité d'entité) et clés étrangères (intégrité référentielle).

🔍 Dans la mesure du possible, il est préférable d'implémenter l'intégrité sous la forme de contrainte car la contrainte fait alors partie intégrante de la structure de la table. Le respect de la contrainte est effectif par toutes les lignes d'informations et la vérification est beaucoup plus rapide.

1. Les valeurs par défaut

Depuis SQL Server 2005, les objets DEFAULT n'ont plus cours et ne doivent pas être mis en place dans le cadre de nouveaux développements. En effet, ce type d'objet n'est pas conforme à la norme du SQL.

Même si les instructions CREATE DEFAULT, DROP DEFAULT, sp_bindefault et sp_unbindefault sont toujours présentes, il est préférable de définir la valeur par défaut lors de la création de la table (CREATE TABLE) ou de passer par une instruction de modification de table (ALTER TABLE). Comme toujours, ces opérations peuvent être exécutées sous forme de script ou par l'intermédiaire de SQL Server Management Studio.

2. Les règles

Afin de proposer une gestion plus uniforme des différents éléments de la base, en généralisant l'utilisation des instructions CREATE, ALTER et DROP, et pour être plus proche de la norme, SQL Server 2008 ne propose plus la gestion des règles en tant qu'objet indépendant. Les contraintes d'intégrité qui pouvaient être exprimées sous forme de règle doivent être définies lors de la création de la table par l'instruction CREATE TABLE. Elles peuvent également être ajoutées/supprimées sur une table existante par l'intermédiaire de l'instruction ALTER TABLE. Pour assurer la continuité des scripts, SQL Server continue d'interpréter correctement les instructions CREATE RULE, DROP RULE, sp_bindrule, sp_unbindrule.

3. La propriété Identity

Cette propriété peut être affectée à une colonne numérique entière, à la création ou à la modification de la table et permet de faire générer, par le système, des valeurs pour cette colonne. Les valeurs seront générées à la création de la ligne, successivement en partant de la valeur initiale spécifiée (par défaut 1) et en augmentant ou diminuant ligne après ligne d'un incrément (par défaut 1).

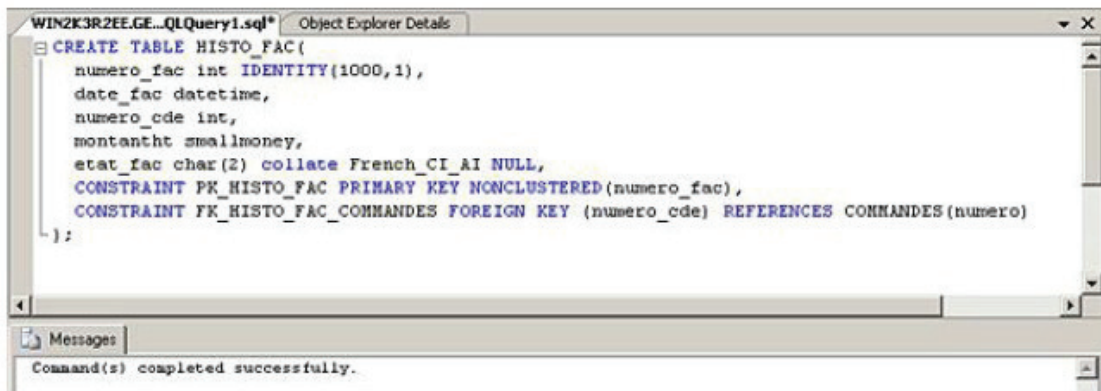
Syntaxe

CREATE TABLE nom (colonne typeentier IDENTITY [(depart, increment)],
...)

Il ne peut y avoir qu'une colonne IDENTITY par table !

La propriété IDENTITY doit être définie en même temps que la colonne à laquelle elle est rattachée. La définition d'une colonne identity peut intervenir dans une commande CREATE TABLE ou bien dans une commande ALTER TABLE.

Exemple



Lors des créations de ligne (INSERT), on ne précisera pas de valeur pour NUMERO_FAC. La première insertion affectera le NUMERO_FAC 1000, la deuxième le NUMERO_FAC 1001, etc. Le mot clé IDENTITYCOL pourra être utilisé dans une clause WHERE à la place du nom de colonne.

La variable globale @@IDENTITY stocke la dernière valeur affectée par une identité au cours de la session courante. La fonction SCOPE_IDENTITY permet d'effectuer le même type de travail mais limite la portée de la visibilité au seul lot d'instructions courant. La fonction IDENT_CURRENT permet quant à elle de connaître la dernière valeur identité générée pour la table spécifiée en paramètre, quelles que soient les sessions.

Pour pouvoir insérer des données sans utiliser la propriété IDENTITY et donc la numération automatique, il faut faire appel à l'instruction IDENTITY_INSERT de la façon suivante :

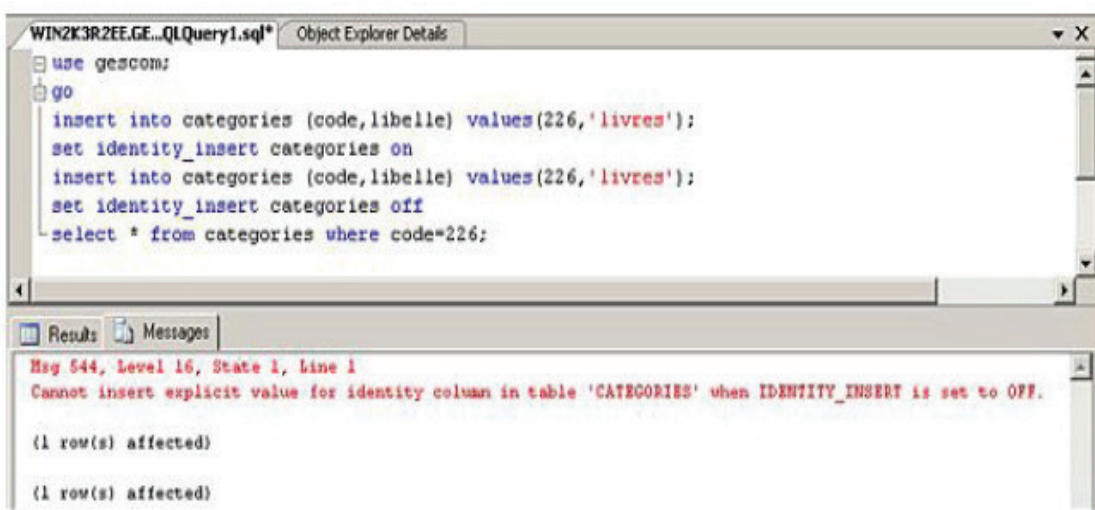
SET IDENTITY_INSERT nom_de_table ON

Le paramètre ON permet de désactiver l'usage de la propriété IDENTITY tandis que la même instruction avec le paramètre OFF réactive la propriété.

Exemple

Dans l'exemple suivant, une nouvelle catégorie est ajoutée. La première insertion se solde par un échec car la propriété **IDENTITY** est active.

Après sa désactivation par l'instruction **SET IDENTITY_INSERT categories ON**, il est possible d'insérer la ligne de données.



The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results/messages pane. The query editor contains the following SQL code:

```
use gescom;
go
insert into categories (code, libelle) values (226, 'livres');
set identity_insert categories on
insert into categories (code, libelle) values (226, 'livres');
set identity_insert categories off
select * from categories where code=226;
```

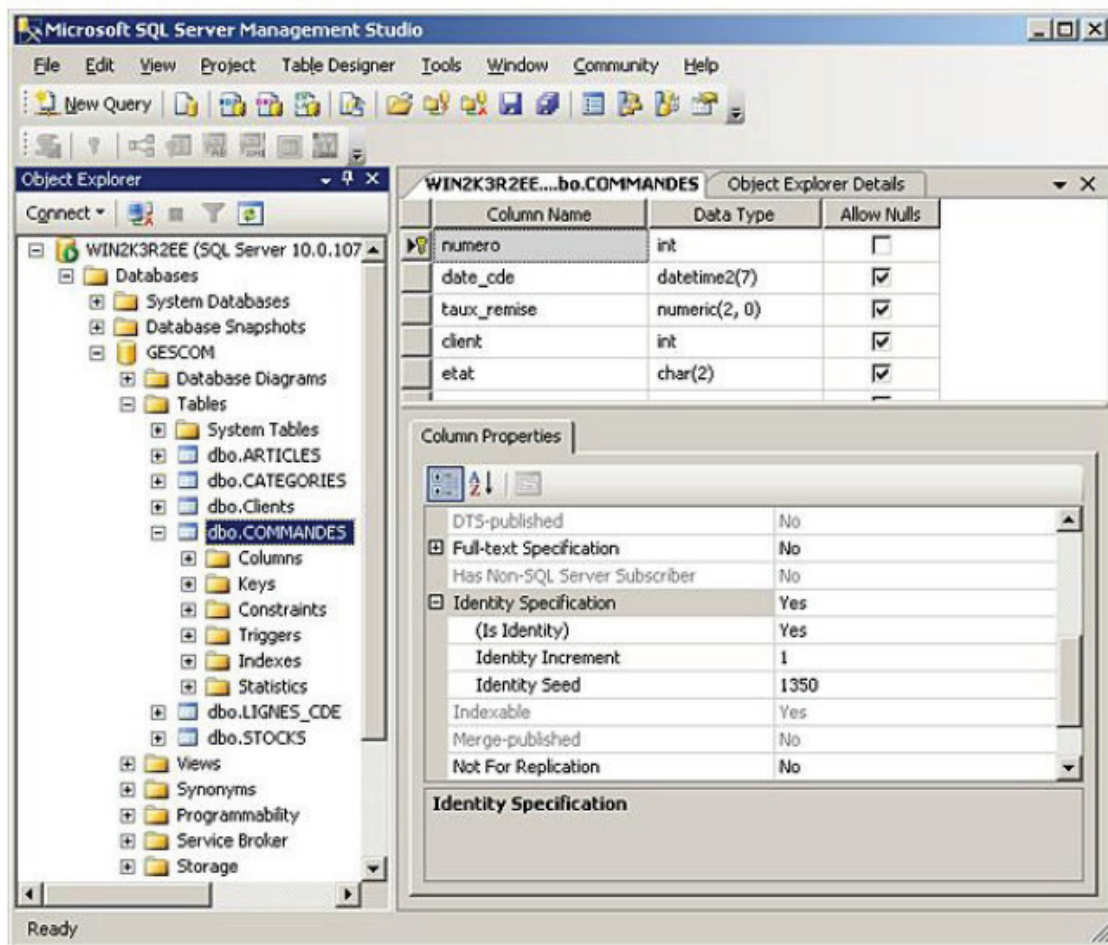
The results/messages pane shows the following error message:

```
Msg 544, Level 16, State 1, Line 1
Cannot insert explicit value for identity column in table 'CATEGORIES' when IDENTITY_INSERT is set to OFF.

(1 row(s) affected)

(1 row(s) affected)
```

Il est possible de définir la propriété identity depuis SQL Server Management Studio en affichant l'écran de modification ou de création d'une table (**Design/Modifier** depuis le menu contextuel associé à la table).



Il est possible d'utiliser les fonctions suivantes pour obtenir plus informations sur les types identités :

- IDENT_INCR pour connaître le pas d'incrément de la valeur identity.
- IDENT_SEED pour connaître la valeur de départ fixée lors de la création du type identity.

Toutes ces fonctions ont pour objectif de permettre au programmeur de mieux contrôler la valeur générée afin de pouvoir la récupérer lorsqu'il s'agit de la clé primaire.

4. Les contraintes d'intégrité

Les contraintes permettent de mettre en oeuvre l'intégrité déclarative en définissant des contrôles de valeur au niveau de la structure de la table elle même.

La définition des contraintes se fait sous forme de script par l'intermédiaire des instructions CREATE et ALTER TABLE. Il est également possible de les définir depuis SQL Server Management Studio.

Il est recommandé de passer, lorsque cela est possible, par des contraintes d'intégrité à la place de déclencheurs de base de données car les contraintes d'intégrités sont normalisées et elles limitent le codage donc le risque d'erreur.

Elles sont intégrées dans la définition de la structure de la table et leur vérification est plus rapide que l'exécution d'un déclencheur.

Syntaxe

ALTER TABLE nom Table

{ADD|DROP} CONSTRAINT nom Contrainte ...[:]

Il est possible d'obtenir des informations sur les contraintes d'intégrités définies en interrogeant **sys.key_constraints** ou en utilisant les procédures stockées, **sp_help** et **sp_helpconstraint**.

Les contraintes seront associées à la table ou à une colonne de la table, en fonction des cas.

Il est possible de vérifier l'intégrité des contraintes au travers de DBCC CHECKCONSTRAINT. Cet utilitaire prend tout son sens lorsque les contraintes d'une table ont été désactivées puis réactivées sans vérification des données dans la table.

a. NOT NULL

SQL Server considère la contrainte de nullité comme une propriété de colonne. La syntaxe est donc :

CREATE TABLE nom table (nom colonne type

[{NULL | NOT NULL}] [? ...])

NOT NULL

Spécifie que la colonne doit être valorisée en création ou en modification.

Il est préférable de préciser systématiquement NULL ou NOT NULL, car les valeurs par défaut de cette propriété dépendent de beaucoup de facteurs :

- Pour un type de données défini par l'utilisateur, c'est la valeur spécifiée à la création du type.
- Les types bit et timestamp n'acceptent que NOT NULL.
- Les paramètres de session ANSI_NULL_DFLT_ON ou ANSI_NULL_DFLT_OFF peuvent être activés par la commande SET.
- Le paramètre de base de données 'ANSI NULL' peut être positionné.

Depuis SQL Server 2005, il est possible de modifier la contrainte de NULL/NOT NULL avec une commande ALTER TABLE pour une colonne qui existe déjà. Bien entendu, les données déjà présentes dans la table doivent respecter ces contraintes.

b. PRIMARY KEY

Cette contrainte permet de définir un identifiant clé primaire, c'est à dire une ou plusieurs colonnes n'acceptant que des valeurs uniques dans la table (règle d'unicité ou contrainte d'entité).

Syntaxe

[CONSTRAINT nom contrainte] PRIMARY KEY[CLUSTERED|NONCLUSTERED]
(nom colonne[,...]) [WITH<\$I[>] FILLFACTOR> FILLFACTOR=x][ON groupe_de_fichiers]

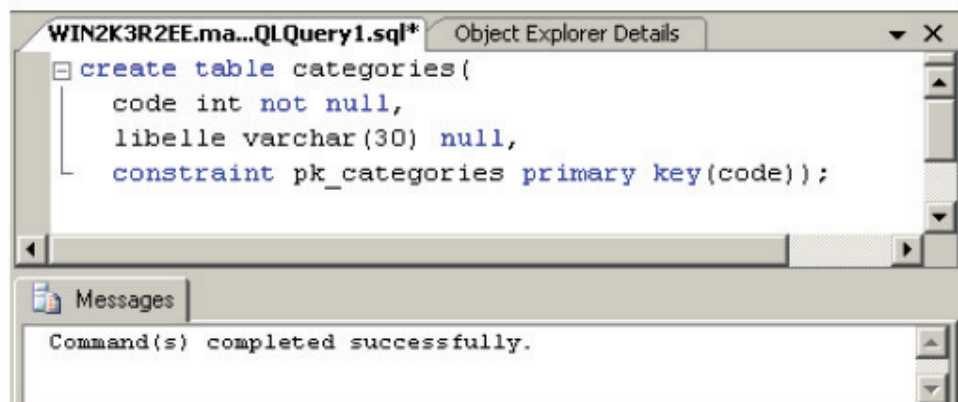
nomcontrainte

Nom permettant d'identifier la contrainte dans les tables système. Par défaut, SQL Server donnera un nom peu facile à manipuler.

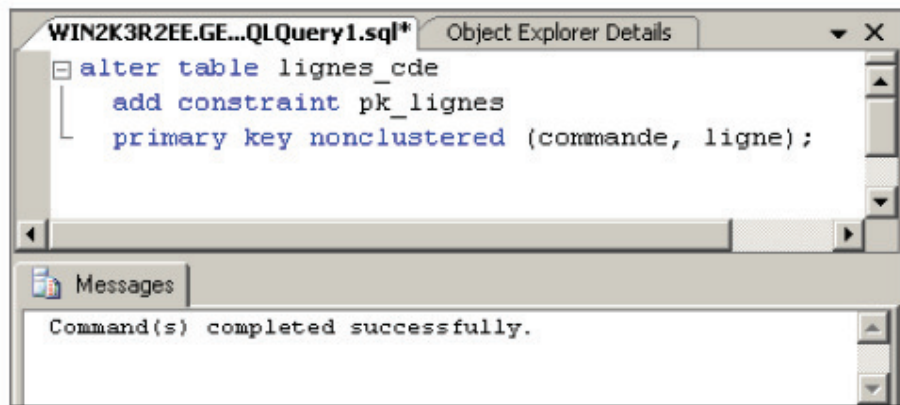
➤ Cette contrainte va automatiquement créer un index unique, ordonné par défaut, du nom de la contrainte, d'où les options NONCLUSTERED et FILLFACTOR. Une clé primaire peut contenir jusqu'à 16 colonnes. Il ne peut y avoir qu'une clé primaire par table. Les colonnes la définissant doivent être NOT NULL.

Exemples

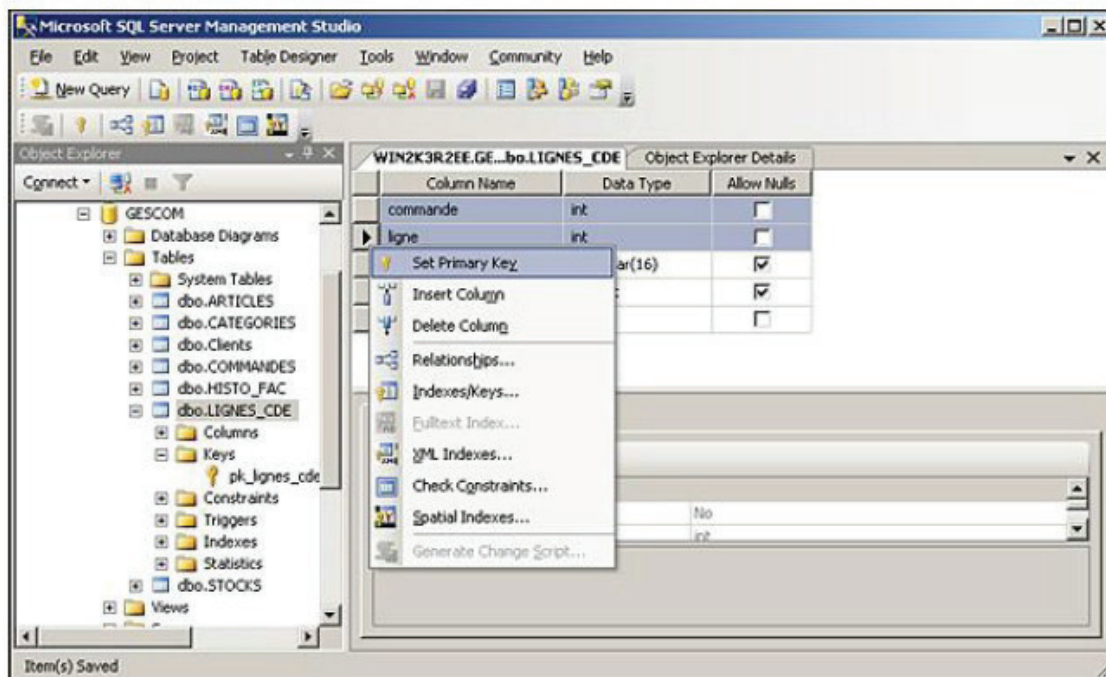
Table catégorie, identifiant CODE_CAT :



Ajout de la clé primaire à la table Lignes_cdes (un index ordonné existe déjà sur Numéro_cde) :



Gestion de la clé primaire par SQL Server Management Studio :



Il n'est pas possible de supprimer la clé primaire si :

- elle est référencée par une contrainte de clé étrangère.
- un index xml primaire est défini sur la table.

c. UNIQUE

Cette contrainte permet de traduire la règle d'unicité pour les autres clés uniques de la table ou identifiants clés secondaires.

Cette contrainte possède les mêmes caractéristiques que PRIMARY KEY à deux exceptions près :

- il est possible d'avoir plusieurs contraintes UNIQUE par table ;
- les colonnes utilisées peuvent être NULL (non recommandé !).

Lors de l'ajout d'une contrainte d'unicité sur une table existante, SQL Server s'assure du respect de cette contrainte par les lignes déjà présentes avant de valider l'ajout de la contrainte.

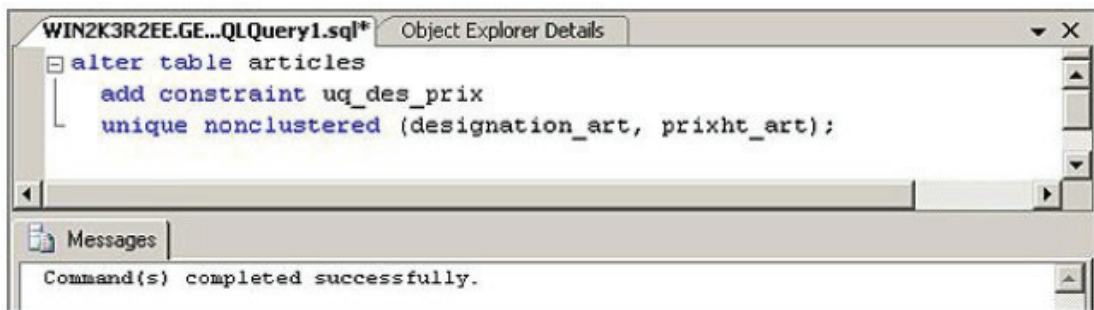
La gestion de cette contrainte est assurée par un index de type UNIQUE. Il n'est pas possible de supprimer cet index par l'intermédiaire de la commande DROP INDEX. Il faut supprimer la contrainte par l'intermédiaire de ALTER TABLE.

Syntaxe

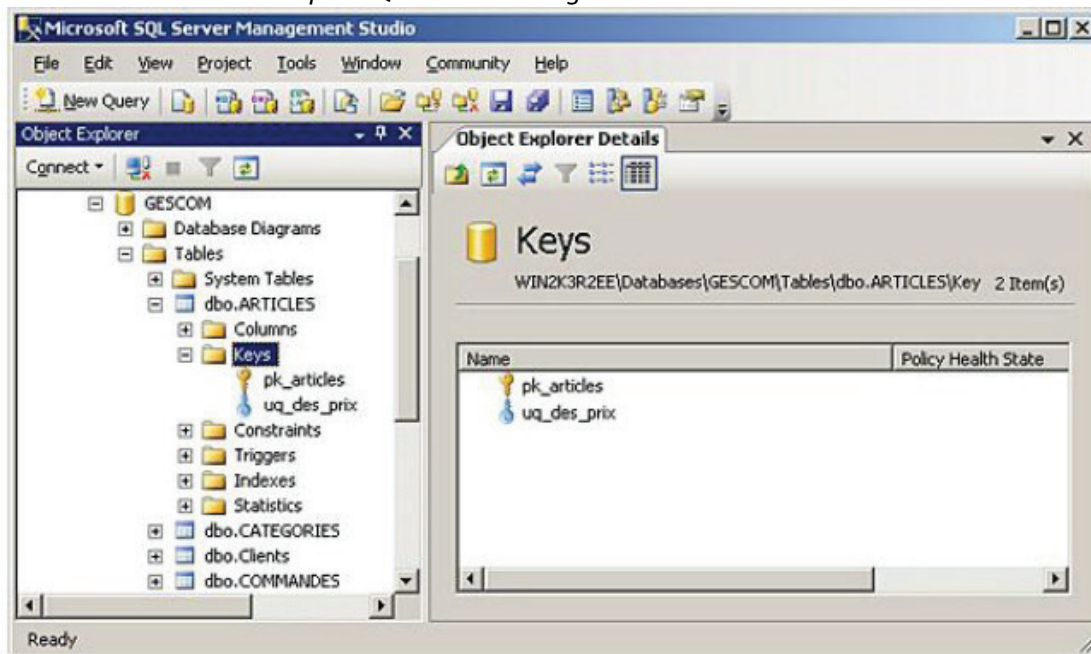
[CONSTRAINT nom contrainte] UNIQUE [CLUSTERED | NONCLUSTERED]
(nom colonne [...]) [WITH FILLFACTOR=x] [ON groupe_de_fichiers]

Exemple

L'association des colonnes Designation et Prixht doit être unique dans la table ARTICLES :



Gestion des clés secondaires par SQL Server Management Studio :



d. REFERENCES

Cette contrainte traduit l'intégrité référentielle entre une clé étrangère d'une table et une clé primaire ou secondaire d'une autre table.

Syntaxe

```
CONSTRAINT nom_contrainte  
[FOREIGN KEY (colonne[,...])]  
REFERENCES table [ ( colonne [<+>,... ) ] ]  
[ ON DELETE { CASCADE | NO ACTION | SET NULL | SET DEFAULT } ]  
[ ON UPDATE { CASCADE | NO ACTION | SET NULL | SET DEFAULT } ]
```

La clause FOREIGN KEY est obligatoire lorsqu'on utilise une syntaxe contrainte de table pour ajouter la contrainte.

L'option de cascade permet de préciser le comportement que doit adopter SQL Server lorsque l'utilisateur met à jour ou tente de supprimer une colonne référencée. Lors de la définition d'une contrainte de référence par les instructions CREATE TABLE ou ALTER TABLE, il est possible de préciser les clauses ON DELETE et ON UPDATE.

NO ACTION

Valeur par défaut de ces options. Permet d'obtenir un comportement identique à celui présent dans les versions précédentes de SQL Server.

ON DELETE CASCADE

Permet de préciser qu'en cas de suppression d'une ligne dont la clé primaire est référencée par une ou plusieurs lignes, toutes les lignes contenant la clé étrangère faisant référence à la clé primaire sont également supprimées. Par exemple, avec cette option, la suppression d'une ligne d'information dans la table des commandes provoque la suppression de toutes les lignes d'information de la table lignes_commande.

ON UPDATE CASCADE

Permet de demander à SQL Server de mettre à jour les valeurs contenues dans les colonnes de clés étrangères lorsque la valeur de clé primaire référencée est mise à jour.

SET NULL

Lorsque la ligne correspondant à la clé primaire dans la table référencée est supprimée, alors la clé étrangère prend la valeur null.

SET DEFAULT

Lorsque la ligne correspondant à la clé primaire dans la table référencée est supprimée, alors la clé étrangère prend la valeur par défaut définie au niveau de la colonne.



Une action en cascade n'est pas possible sur les tables munies d'un déclencheur instead of.

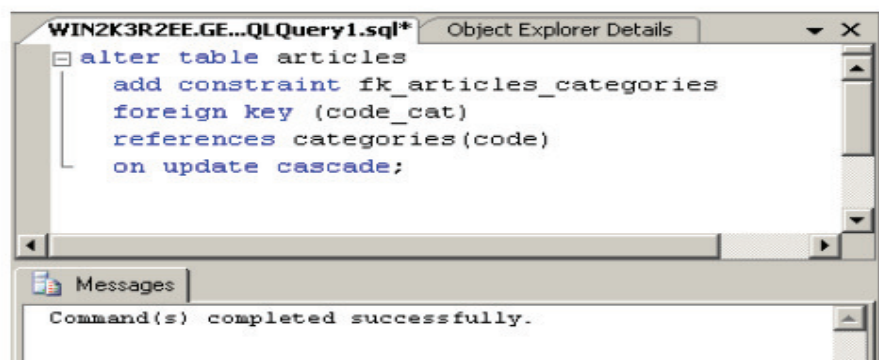
Toute référence circulaire est interdite dans les déclenchements des cascades.

La contrainte de référence ne crée pas d'index. Il est recommandé de le créer par la suite.

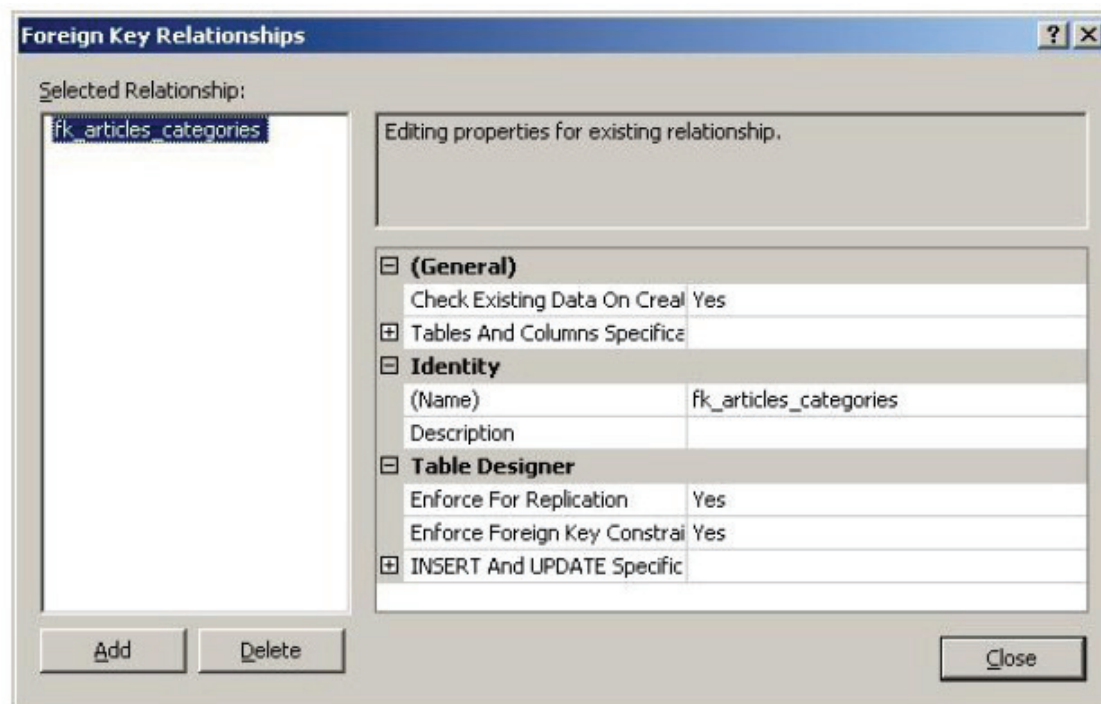
Bien que SQL Server ne comporte pas de limite maximale en ce qui concerne le nombre maximal de contraintes de clé étrangère définies sur chaque table, il est recommandé par Microsoft de ne pas dépasser les 253. Cette limite est à respecter en ce qui concerne le nombre de clés étrangères définies sur la table et le nombre de clés étrangères qui font référence à la table. Au delà de cette limite, il peut être intéressant de se pencher à nouveau sur la conception de la base pour obtenir un schéma plus optimum.

Exemple

Création de la clé étrangère `code_cat` dans la table `ARTICLES` :



Gestion des clés étrangères par SQL Server Management Studio :



e. DEFAULT

La valeur par défaut permet de préciser la valeur qui va être positionnée dans la colonne si aucune information n'est précisée lors de l'insertion de la ligne.

Les valeurs par défaut sont particulièrement utiles lorsque la colonne n'accepte pas les valeurs NULL car elles garantissent l'existence d'une valeur.

Il faut toutefois conserver à l'esprit que la valeur par défaut est utilisée uniquement lorsqu'aucune valeur n'est précisée pour la colonne dans l'instruction INSERT. Il n'est pas possible de compléter ou d'écraser une valeur saisie par l'utilisateur. Pour réaliser ce type d'opération, il est nécessaire de développer un déclencheur de base de données.

Une valeur par défaut peut être définie pour toutes les colonnes à l'exception des colonnes de type timestamp ou bien celles qui possèdent un type identity.

Syntaxe

[CONSTRAINT Nom contrainte] DEFAULT valeur

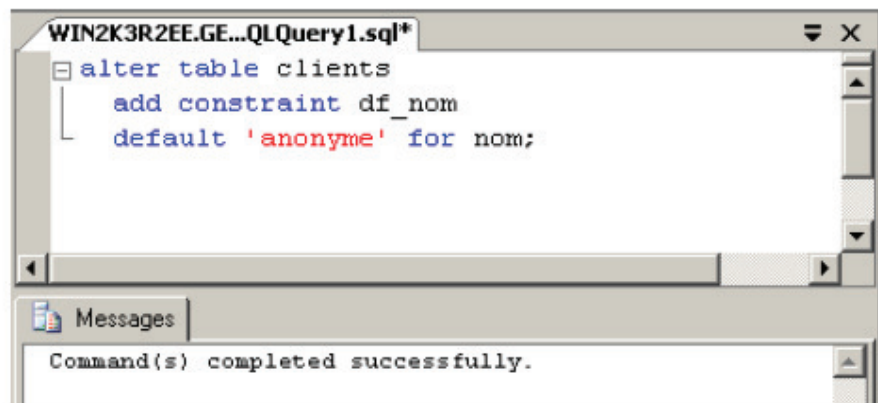
[FOR nom colonne]

valeur

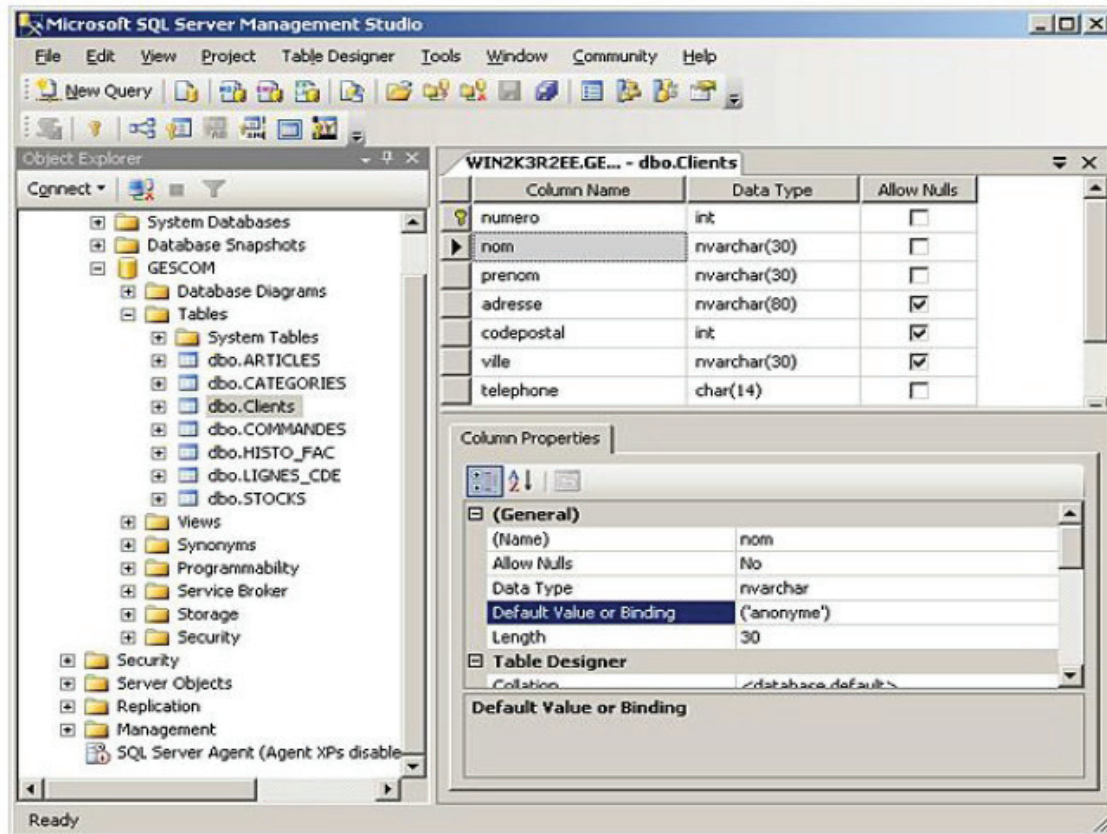
La valeur doit être exactement du même type que celui sur lequel est définie la colonne. Cette valeur peut être une constante, une fonction scalaire (comme par exemple : USER, CURRENT_USER, SESSION_USER, SYSTEM_USER...) ou bien la valeur NULL.

Exemple

Valeur par défaut pour le Nom du client :



Définition d'une valeur par défaut depuis SQL Server Management Studio :



f. CHECK

La contrainte de validation ou contrainte CHECK, permet de définir des règles de validation mettant en rapport des valeurs issues de différentes colonnes de la même ligne. Ce type de contrainte permet également de s'assurer que les données respectent un format précis lors de leur insertion et mise à jour dans la table. Enfin, par l'intermédiaire d'une contrainte de validation, il est possible de garantir que la valeur présente dans la colonne appartient à un domaine précis de valeurs.

Syntaxe

[CONSTRAINT Nom contrainte] CHECK [NOT FOR REPLICATION]
(expression booléenne)

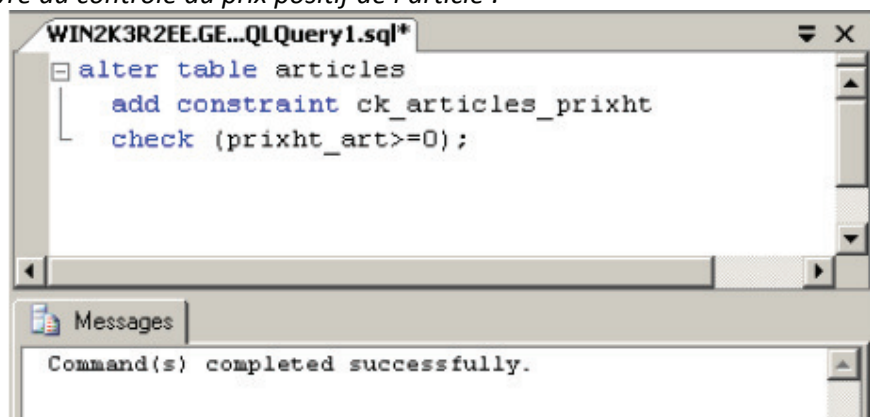
NOT FOR REPLICATION

Permet d'empêcher l'application de la contrainte lors de la réplication.

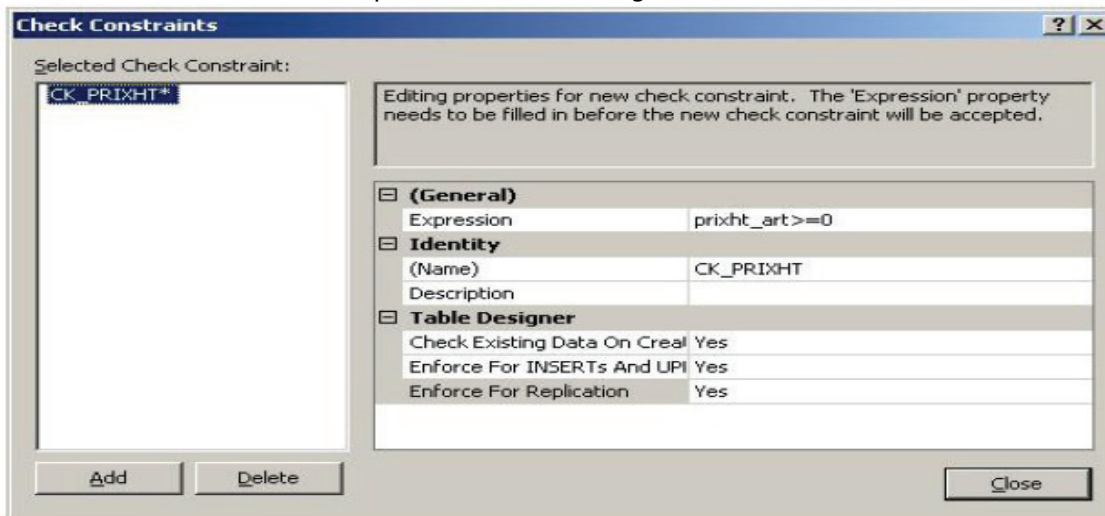
➤ La contrainte CHECK est automatiquement associée à la colonne spécifiée dans l'expression de la condition.

Exemple

Mise en oeuvre du contrôle du prix positif de l'article :



Gestion des contraintes CHECK par SQL Server Management Studio :



5. Gérer les index

Utilisation des index ou pourquoi indexer ?

L'objectif des index est de permettre un accès plus rapide à l'information dans le cadre des extractions (SELECT) mais également dans le cadre des mises à jour (INSERT, UPDATE, DELETE) en réduisant le temps nécessaire à la localisation de la ligne. Cependant, les index vont être coûteux dans le cas de mise à jour de la valeur contenue dans la colonne indexée.

Une bonne stratégie d'indexation doit prendre en considération ces différents points et peut déjà en déduire deux règles :

- Il est préférable d'avoir trop peu d'index que trop d'index. Dans le cas de multiples index, les gains en accès à l'information sont annulés par les temps nécessaires pour mettre à niveau les index.
- Les index doivent être le plus "large" possible afin de pouvoir être utilisés par plusieurs requêtes.

Enfin, il faut s'assurer que les requêtes utilisent bien les index qui sont définis.

Les requêtes doivent également être écrites pour manipuler le minimum d'informations de la façon la plus explicite possible.

Par exemple, dans le cas d'une projection, il est préférable de lister les colonnes pour lesquelles l'information doit être affichée à la place du caractère générique *.

Pour les restrictions, il est préférable de faire des comparaisons entre constantes et la valeur contenue dans une colonne.

Par exemple, si la table des Articles contient le prix HT de chaque article, pour extraire la liste des articles dont le prix TTC est inférieur ou égal à 100 €, il est préférable d'écrire la condition suivante $\text{prixht} \leq 100 / 1.196$ à la place de $\text{prixht} * 1.196 \leq 100$. En effet, dans le second cas, le calcul est effectué pour chaque article, tandis que dans le premier, le calcul est fait une fois pour toute. Il faut également prendre en compte le fait que les données sont stockées dans les index et donc qu'ils vont occuper un espace disque non négligeable. Le niveau feuille d'un index ordonné (clustered) contient l'ensemble des données. Pour un index non ordonné, le niveau feuille de l'index contient une référence directe vers la ligne d'information liée à la clé de l'index.

Les autres niveaux de l'index sont utilisés pour naviguer dans l'index et arriver ainsi très rapidement à l'information. Il est possible d'ajouter de l'information au niveau des feuilles de l'index, sans que ces colonnes soient prises en compte par l'index. Cette technique est pratique dans le cas de la définition d'index qui couvre des requêtes. Par exemple, il est nécessaire d'extraire la liste des codes postaux et des villes de la table des clients. Pour cela, un index non ordonné est défini par rapport à la colonne des codes postaux, et la colonne qui

représente le nom de la ville est ajouté au niveau feuille. Ainsi, l'index couvre la requête qui est capable de produire le résultat sans faire d'accès à la table.

Qu'est ce qu'un index ?

La notion d'index est déjà connue de tous. En effet, chacun a déjà utilisé l'index d'un livre pour aboutir directement à la page ou aux pages d'informations recherchées. Peut être avez vous d'ailleurs utilisé l'index de ce livre pour aboutir directement à cette explication en parcourant l'index à la recherche du mot clé "index".

Si pour l'index, derrière chaque mot clé n'apparaît qu'un seul numéro de page, on parle alors d'index unique.

Les index que SQL Server propose de mettre en place sont très proches des index que l'on peut trouver dans les livres.

En effet, il est possible de parcourir l'ensemble de l'index pour retrouver toutes les informations, comme il est possible de lire un livre à partir de l'index, au lieu de suivre l'ordre proposé par la table des matières.

Il est également possible d'utiliser l'index pour accéder directement à une information précise. Afin de garantir des temps d'accès à l'information homogènes, SQL Server structure l'information sous forme d'arborescence autour de la propriété indexée. C'est d'ailleurs la démarche que l'on adopte lorsque l'on parcourt un index en effectuant une première localisation de l'information par rapport au premier caractère, puis un parcours séquentiel afin de localiser le mot clé recherché.

Maintenant, imaginez un livre sur lequel il est possible de définir plusieurs index : par rapport aux mots clés, par rapport aux thèmes, par rapport au type de manipulations que l'on souhaite faire... Cette multitude d'index, c'est ce que propose SQL Server en donnant la possibilité de créer plusieurs index sur une table.

Parmi tous les index du livre, un en particulier structure le livre : c'est la table des matières, qui peut être perçue comme un index sur les thèmes. De même, SQL Server propose de structurer physiquement les données par rapport à un index. C'est l'index CLUSTERED.

Organiser ou non les données ?

SQL Server propose deux types d'index : les index organisés, un au maximum par table, qui réorganisent physiquement la table et les index non organisés.



La définition ou bien la suppression d'un index non organisé n'a aucune influence sur l'organisation des données dans la table. Par contre, la définition ou la suppression d'un index organisé aura des conséquences sur la structure des index non organisés.

Table sans index organisé

Si une table possède uniquement ce type d'index, alors les informations sont stockées au fil de l'eau sans suivre une organisation quelconque.

Ce choix est particulièrement adapté, lorsque :

- la table stocke de l'information en attendant un partitionnement,
- les informations vont être tronquées,
- des index sont fréquemment créés ou bien supprimés,
- un chargement en bloc de données a lieu,
- les index sont créés après le chargement des données, et leur création peut être parallélisée,
- les données sont rarement modifiées (UPDATE) afin de conserver une structure solide,
- l'espace disque utilisé par l'index est minimisé, ce qui permet de définir des index à moindre coût.

Cette solution est un choix performant pour l'indexation d'une table non présente sur un serveur OLTP, comme un serveur dédié à l'analyse des informations.

Les index organisés

Sur chaque table, il est possible de définir un et un seul index organisé. Ce type d'index permet d'organiser physiquement les données contenues dans la table selon un critère particulier. Par exemple, un index organisé peut être défini sur la clé primaire.

Ce type d'index est coûteux en temps et en espace disque pour le serveur lors de sa construction ou de sa reconstruction.

Si ce type d'index est défini sur une table déjà valorisée, sa construction sera longue. Elle sera d'ailleurs d'autant plus longue si des index non ordonnés existent déjà.

Idéalement, et afin d'éviter une maintenance trop coûteuse de l'index ordonné, il est défini sur une colonne qui contient des données statiques et qui occupe un espace limité, comme la clé primaire par exemple.

La définition d'un index organisé sur une colonne non stable, comme le nom d'une personne ou son adresse, conduit irrémédiablement à une dégradation significative des performances.

Les index non organisés

Suivant qu'elles soient définies sur une table munie ou non d'un index organisé, les feuilles de l'index non organisé ne font pas référence à la ou les lignes d'informations de la même façon. Tout d'abord, dans le cas où la table ne possède pas d'index organisé, le RID (Row Identifier) de la ligne d'information est stocké au niveau des feuilles de l'index. Ce RID correspond à l'adresse physique (au sens SQL Server) de la ligne. Si au contraire la table possède un index organisé, alors, au niveau de la feuille de l'index non organisé, est stockée la clé de la ligne d'information recherchée. Cette clé correspond au critère de recherche de l'index organisé.

Les index non organisés sont à privilégier pour définir un index qui couvre une ou plusieurs requêtes. Avec ce type d'index, la requête trouve dans l'index l'ensemble des informations dont elle a besoin et évite ainsi un accès inutile à la table, puisque seul l'index est manipulé. Les performances sont alors considérablement améliorées car le volume de données manipulé est beaucoup plus léger.

Bien entendu, chaque requête peut être optimisée à l'aide de cette technique, mais ce n'est pas non plus l'objectif recherché car la multitude d'index serait coûteuse à maintenir.

Index et calcul d'agrégat

Pour les tables volumineuses, des index doivent être mis en place afin de s'assurer que les requêtes courantes ne provoquent pas un balayage complet de table mais s'appuient simplement sur les index. Ce point est particulièrement important pour les calculs d'agrégat qui nécessitent une opération de tri avant de pouvoir effectuer le calcul. Si un index permet de limiter le tri à effectuer, alors le gain de performance est très net.

Toujours dans cette optique d'optimisation des performances lors de l'accès aux données, il est possible de définir des index sur des colonnes d'une vue, même si le résultat présent dans la colonne est le résultat d'un calcul d'agrégat.

L'index défini sur une vue est limité à 16 colonnes et 900 octets de données pour chaque entrée de l'index.

Contrairement à l'inclusion de colonnes non indexées dans les feuilles de l'index, les index définis sur une vue peuvent contenir le résultat d'un calcul d'agrégat.

a. Créer un index

Un index peut être créé n'importe quand, qu'il y ait ou non des données dans la table.

Cependant, dans le cas où des données doivent être importées, il est préférable d'importer les données en premier puis de définir les index. Dans le cas contraire (les index sont définis avant une importation majeure de données), il est nécessaire de reconstruire les index afin de garantir une répartition équilibrée des informations dans l'index.

Syntaxe

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX nom_index  
ON { nom_table | nom_vue } ( colonne [ ASC | DESC ] [ ,...n ] )  
[ INCLUDE (colonne[ ,...n ] ) ]  
[ WITH [ PAD_INDEX = {ON | OFF} ]  
[,FILLFACTOR = x]  
[,IGNORE_DUP_KEY = {ON | OFF} ]  
[,DROP_EXISTING = {ON | OFF} ]  
[,ONLINE = {ON | OFF} ]  
[,STATISTICS_NORECOMPUTE = {ON | OFF} ]  
[,SORT_IN_TEMPDB = {ON | OFF} ]  
[ ON groupe_fichier ]
```

UNIQUE

Précise que la ou les colonnes indexées ne pourront pas avoir la même valeur pour plusieurs lignes de la table.

CLUSTERED ou NONCLUSTERED

Avec un index CLUSTERED (ordonné), l'ordre physique des lignes dans les pages de données est identique à l'ordre de l'index. Il ne peut y en avoir qu'un par table et il doit être créé en premier. La valeur par défaut est NONCLUSTERED.

INCLUDE

Avec cette option, il est possible de dupliquer l'information pour inclure une copie des colonnes spécifiée en paramètre directement dans l'index. Cette possibilité est limitée aux index non organisés et les informations sont stockées au niveau feuille. Il est possible d'inclure dans l'index de 1 à 1026 colonnes de n'importe quel type hormis varchar(max), varbinary(max) et nvarchar(max). Cette option INCLUDE permet de définir des index qui couvrent la requête, c'est à dire que la requête va parcourir uniquement l'index pour trouver la totalité des besoins qui lui sont nécessaires.

FILLFACTOR=x

Précise le pourcentage de remplissage des pages d'index au niveau feuille. Cela permet d'améliorer les performances en évitant d'avoir des valeurs d'index consécutives qui ne seraient plus physiquement contiguës. La valeur par défaut est 0 ou fixée par sp_configure. Pour x = 0, le niveau feuille est rempli à 100%, de l'espace est réservé au niveau non feuille.

Pour x entre 1 et 99, le pourcentage de remplissage au niveau feuille est x%, de l'espace est réservé au niveau non feuille. Pour x = 100 les niveaux feuille et non feuille sont remplis.

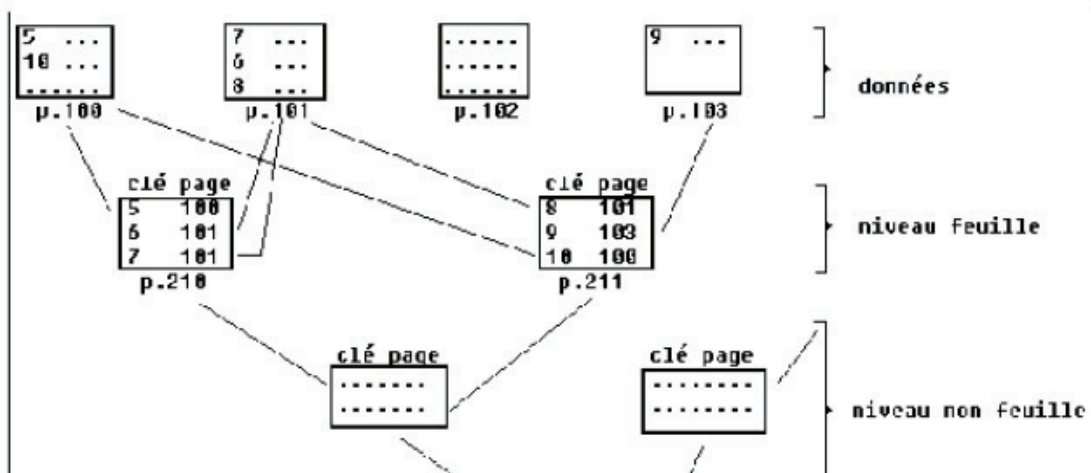
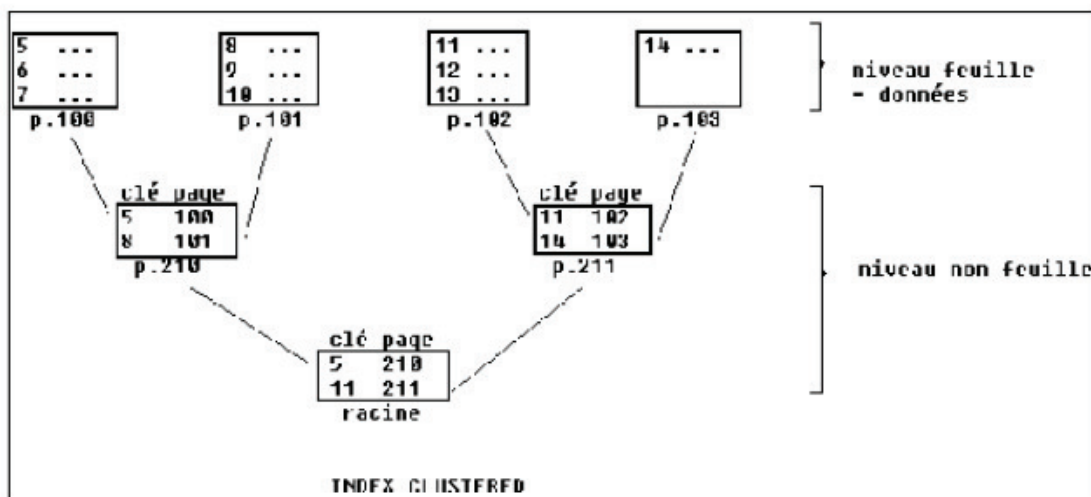
PAD_INDEX

Précise le taux de remplissage des pages non feuille de l'index. Cette option n'est utilisable qu'avec FILLFACTOR dont la valeur est reprise.

Groupe fichier

Groupe de fichiers sur lequel est créé l'index.

Les index partitionnés ne sont pas pris en compte à ce niveau. Les éléments de syntaxe portent uniquement sur les index définis entièrement sur un et un seul groupe de fichiers.



IGNORE_DUP_KEY

Cette option autorise des entrées doubles dans les index uniques. Si cette option est levée, un avertissement est généré lors de l'insertion d'un doublon et SQL Server ignore l'insertion de ligne. Dans le cas contraire, une erreur est générée.

DROP_EXISTING

Précise que l'index déjà existant doit être supprimé.

ONLINE

Lorsque cette option est activée (ON) lors de la construction de l'index, les données de la table restent accessibles en lecture et en modification pendant la construction de l'index. Cette option est disponible à partir de SQL Server 2005 et elle est désactivée par défaut.

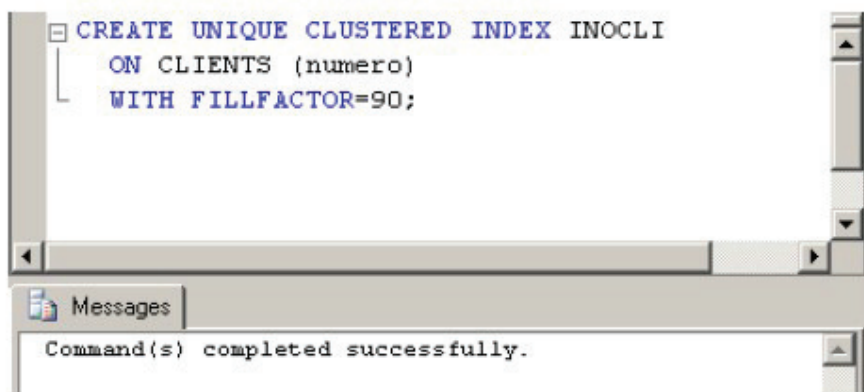
STATISTICS_NORECOMPUTE

Les statistiques d'index périmés ne sont pas recalculées, et il faudra passer par la commande **UPDATE STATISTICS** pour remettre à jour ces statistiques.

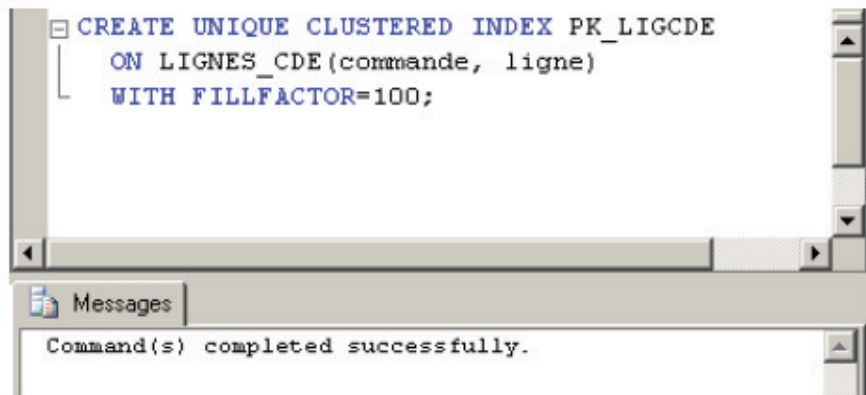
Si le serveur sur lequel s'exécute SQL Server possède plusieurs processeurs, alors la création d'index peut être parallélisée afin de gagner du temps pour la construction d'index sur des tables de grandes dimensions. La mise en place d'un plan d'exécution parallèle pour la construction d'un index prend en compte le nombre de processeurs du serveur fixé par l'option de configuration max degree of parallelism (sp_configure) et le nombre de processeurs n'étant pas déjà trop chargés par des threads SQL Server.

Exemples

Création d'un index ordonné :



Création d'un index ordonné sur des données déjà triées :



b. Supprimer un index

Seuls les index définis avec l'instruction CREATE INDEX peuvent être supprimés avec DROP INDEX. Un index peut être supprimé lorsque sa présence ne permet pas un gain de performances significatif en comparaison du coût de maintenance.

Si la suppression intervient dans le cadre d'une reconstruction de l'index, il est alors préférable d'activer l'option DROP_EXISTING de l'instruction CREATE INDEX car elle offre de meilleures performances.

Syntaxe

DROP INDEX nomIndex ON nomTable;

c. Reconstruire un index

La commande DBCC DBREINDEX est encore disponible pour des raisons de compatibilité. Il est préférable d'utiliser la commande ALTER INDEX qui permet de maintenir les index.

La commande ALTER INDEX permet de reconstruire un index en particulier ou tous les index associés à une table. Lors de la reconstruction de l'index, il est possible de préciser le facteur de remplissage des pages feuilles.

Syntaxe

ALTER INDEX { nomIndex | ALL }

```
ON nomTable  
REBUILD  
[WITH](  
[PAD_INDEX = { ON | OFF },]  
[FILLFACTOR = x ,]  
[SORT_IN_TEMPDB = { ON | OFF },]  
[IGNORE_DUP_KEY = { ON | OFF },]  
[STATISTICS_NORECOMPUTE = { ON | OFF }]  
[ONLINE = { ON | OFF },]  
[;]
```

FILLFACTOR

Permet de spécifier le pourcentage de remplissage des pages au niveau feuille de l'index.

PAD_INDEX

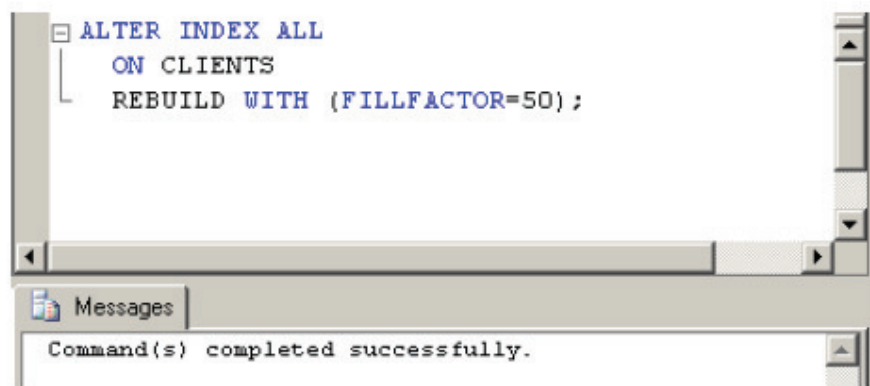
Permet de reporter dans les pages non feuilles de l'index, le même niveau de remplissage que celui précisé par l'intermédiaire de FILLFACTOR.

Les autres options de l'instruction ALTER INDEX possèdent la même signification que celles utilisées avec l'instruction CREATE INDEX. Il est à remarquer que l'option ONLINE prend tout son sens dans ce cas précis car elle permet de reconstruire l'index en laissant les utilisateurs travailler avec les données de la table sous jacente.

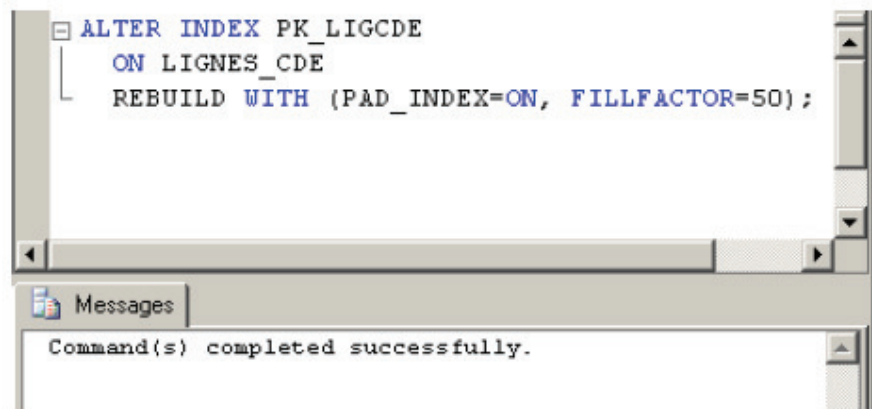
Certes les opérations seront plus lentes mais elles ne seront pas bloquées. Si la reconstruction d'index est planifiée sur un moment de faible utilisation du serveur elle peut même être transparente pour les quelques utilisateurs qui travaillent sur la base à ce moment.

L'exemple ci dessous

illustre la reconstruction de tous les index d'une table :



*Ce second exemple illustre la reconstruction d'un index en spécifiant une valeur pour l'option **FILLFACTOR**, ainsi que la prise en compte du facteur de remplissage dans les fonctions non feuilles.*



d. Mettre à jour les statistiques

SQL Server utilise des informations sur la distribution des valeurs de clés pour optimiser les requêtes. Ces informations doivent être mises à jour après des modifications importantes de données.

Bien que la procédure manuelle de création et de mise à jour des statistiques soit exposée ici, il est très nettement préférable de configurer la base pour une création et une mise à jour automatique des statistiques. En effet, trop souvent la dégradation des performances d'un serveur est due en partie ou en totalité à des statistiques non mises à jour.

Syntaxe

UPDATE STATISTICS nomtable [,nomindex]

[WITH {FULLSCAN|SAMPLE n {PERCENT|ROWS}|RESAMPLE}]

Si le nom d'index est omis, tous les index sont pris en compte.

FULLSCAN

Les statistiques sont créées à partir d'un balayage complet de la table, soit 100% des lignes.

SAMPLE n{PERCENT|ROWS}

Les statistiques sont établies à partir d'un échantillon représentatif des informations contenues dans la table. Cet échantillon peut être exprimé en pourcentage ou bien en nombre de lignes. Si la taille de l'échantillon précisé n'est pas suffisante, SQL Server corrige de lui-même cette taille pour s'assurer d'avoir parcouru environ 1000 pages de

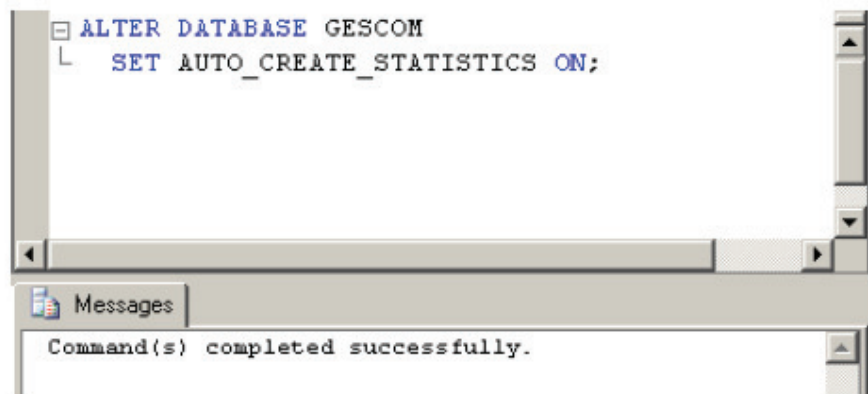
données. C'est le mode d'échantillonnage de statistiques par défaut.

RESAMPLE

Permet de redéfinir les statistiques à partir d'un nouvel échantillonnage.

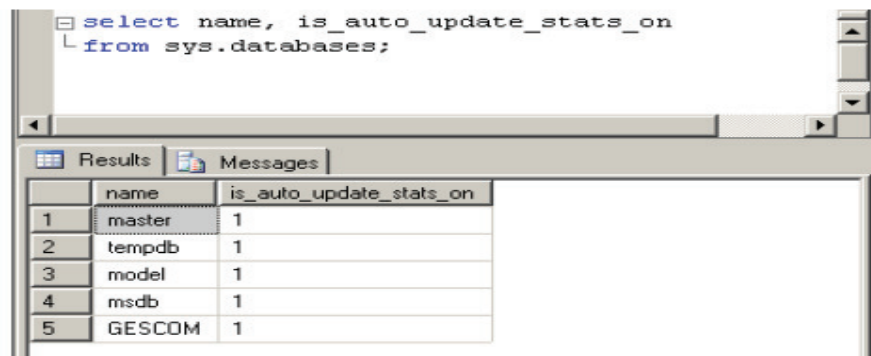
Les statistiques peuvent également être mises à jour de façon automatique. Cette option doit être définie, soit lors de la construction de la base à l'aide de la commande **ALTER DATABASE**, soit en utilisant la procédure stockée **sp_autostats**.

Configuration de la base pour une mise à jour automatique des statistiques d'index :



En mode automatique, c'est le moteur qui se charge de calculer les statistiques manquantes, de maintenir à jour les statistiques en fonction des opérations faites sur les données, mais aussi de supprimer les statistiques inutiles.

Il est possible de savoir si une base est configurée en mise automatique des statistiques en interrogeant la colonne **is_auto_update_stats_on** de la table **sys.databases** ou bien en visualisant la valeur de la propriété **IsAutoUpdateStatistics** de la fonction **databasepropertyex**.



Création des statistiques de tous les index sur les données non système de la base de données GESCOM :



e. Informations sur les index

Des informations concernant la structure des index peuvent être obtenues par les procédures stockées **sp_help** ou **sp_helpindex**.

Des informations concernant la taille et la fragmentation des tables et des index peuvent être obtenues par l'intermédiaire de la fonction **sys.dm_db_index_physical_stats**.

Syntaxe

```
dm_db_index_physical_stats(idBase | NULL,  
idObjet | NULL,  
idIndex | NULL | 0,  
numeroPartition | NULL ,  
mode | NULL | DEFAULT)
```

idBase

Identifiant de la base de données. Il est possible d'utiliser la fonction **db_id()** pour connaître celui de la base



courante. La valeur NULL prend en compte l'ensemble des bases définies sur le serveur et implique d'utiliser la valeur NULL pour l'identifiant de l'objet, de l'index et de la partition.

idObjet

Identifiant de l'objet (table ou vue) sur lequel on souhaite des informations. Cet identifiant peut être obtenu en faisant appel à la fonction **object_id()**. L'utilisation de la valeur NULL permet de signaler que l'on souhaite des informations sur toutes les tables et vues de la base courante. Cela implique également d'utiliser la valeur NULL pour l'option

idIndex et numeroPartition

idIndex

Identifiant de l'index à analyser. Si la valeur NULL est spécifiée alors l'analyse porte sur la totalité des index de la table.

numeroPartition

Numéro de la partition concernée. Si la valeur NULL est spécifiée alors toutes les partitions sont prises en compte.

mode

Précise le mode d'obtention des informations : DEFAULT, NULL, LIMITED ou DETAILED. La valeur par défaut (NULL) correspond à LIMITED

La procédure stockée **sp_spaceused** permet de connaître l'espace disque utilisé par les index.

La fonction **INDEXPROPERTY** permet d'obtenir les différentes informations relatives aux index.