# SQL Server 2008 Express

**Administrez et développez vos bases de données**

Jérôme GABILLAUD


## Résumé

Ce livre sur **SQL Server 2008 Express** s'adresse à toute personne souhaitant travailler efficacement avec cette base de données performante qu'est la version gratuite de SQL Server 2008 (anciennement nommée MSDE). Il présente les différents éléments nécessaires à son administration ainsi que l'ensemble des manipulations à réaliser par l'administrateur, depuis **l'installation** jusqu'aux opérations de **sauvegarde** et de **restauration**, en passant par la gestion de **l'espace disque**, la gestion des **utilisateurs**. Les différentes opérations sont réalisées depuis SQL Server Management Studio et en Transact SQL.
Ce livre permet également de détailler **l'ensemble des instructions** nécessaires à la définition des **tables** ainsi qu'à la **manipulation des données** : les différentes instructions SQL et Transact SQL sont présentées et illustrées afin de bien comprendre l'intérêt des différentes fonctionnalités exposées.
Des éléments sont en téléchargement sur cette page.

## L'auteur

Ingénieur en Informatique pour l'Industrie, consultant, **Jérôme Gabillaud** est également responsable pédagogique dans un grand centre de formation informatique. Spécialiste des systèmes d'accès aux données Microsoft ou Oracle, il est déjà auteur de nombreux ouvrages sur ce sujet, reconnus pour leurs qualités techniques et pédagogiques.

# Introduction

SQL Server est le système de gestion de données relationnelle de Microsoft. Ce moteur est disponible sous différentes formes (éditions) avec différents contrats de licence mais également plus ou moins de fonctionnalités. Parmi ces éditions, ce livre va porter sur l'édition Express de SQL Server 2008.

Cette édition de SQL Server est bien connue des développeurs car elle est intégrée à Visual Studio depuis de nombreuses années. En effet, le terme "Express" n'est apparu qu'avec la version 2005 de SQL Server, auparavant ce produit s'intitulait MSDE (*Microsoft SQL Server Desktop Engines*).

Les avantages de SQL Server Express sont nombreux mais le fait qu'il soit possible de l'installer librement constitue un avantage décisif. SQL Server Express représente donc une très bonne solution à adopter par des applications de taille modérée, qu'elles soient professionnelles ou non. En effet, SQL Server Express, en plus d'être gratuit, est composé du moteur SQL Server. Adopter SQL Server Express dans le cadre d'une application offre la certitude d'héberger les données de façon optimum. De plus, il est possible de migrer vers une autre édition de SQL Server et ceci de façon transparente pour les applications travaillant avec les données.

Bien entendu, cette gratuité de produit s'accompagne de quelques limites :

- La taille de chaque base est limitée à 4 Go maximum.
- Le serveur SQL Server Express ne sait pas exploiter plus de 1 Go de mémoire vive même si le serveur dispose d'une quantité supérieure.

Toutefois, SQL Server 2008 Express est un moteur de base de données pleinement fonctionnel et qui bénéficie des derniers apports de SQL Server 2008 comme le Power Shell, les types de données spatiaux, les nouveaux types de données de gestion des dates et heures…

SQL Server Management Studio reste l'outil principal de travail, que ce soit pour l'administrateur ou bien pour le développeur d'applications. Il est possible d'administrer de façon graphique, toutes les tâches peuvent également être réalisées en utilisant des scripts Transact SQL. Chaque solution possède ses avantages et ses inconvénients. C'est pourquoi les deux solutions sont exposées de façon quasi-systématique dans ce livre. Pour les syntaxes Transact SQL, seules les options les plus courantes seront précisées. L'objectif n'est pas de refaire la documentation mais de présenter au mieux les différentes instructions.

Le composant SQL Server Management Studio n'est pas intégré en tant que tel à l'édition Express de SQL Server 2008. Pour que ce produit soit installé, il est nécessaire d'utiliser l'édition Express Advanced qui intègre la console d'administration SQL Server Management Studio. C'est d'ailleurs le seul écart entre ces deux éditions.

SQL Server utilise sa propre structure de base de données pour stocker toutes les informations relatives à sa propre gestion. Ces informations sont conservées dans les tables dites système. Toutefois, comme la structure de ces tables est amenée à être modifiée lors d'un changement de version, il est recommandé de ne pas interroger directement ces tables mais d'utiliser les vues disponibles dans le schéma sys ou bien INFORMATION_SCHEMA. L'utilisation de ces vues dans des requêtes d'extraction permettra de lire les informations conservées dans le dictionnaire des données.

# Présentation de SQL Server

Le but de ce chapitre est d'acquérir un aperçu de SQL Server dans son ensemble, à savoir :

- comprendre la notion de SGBDR et le mode de fonctionnement client/serveur,
- présenter les composants de SQL Server et les plates-formes d'exécution,
- présenter l'architecture d'administration et de programmation,
- présenter la notion de base de données et les bases installées sur le serveur SQL.

SQL Server est un SGBDR (*Système de Gestion de Base de données Relationnelle*) entièrement intégré à Windows, ce qui autorise de nombreuses simplifications au niveau de l'administration, tout en offrant un maximum de possibilités.

## 1. Qu'est-ce qu'un SGBDR ?

SQL Server est un Système de Gestion de Base de Données Relationnelle (SGBDR), ce qui lui confère une très grande capacité à gérer les données tout en conservant leur intégrité et leur cohérence.

SQL Server est chargé de :

- stocker les données,
- vérifier les contraintes d'intégrité définies,
- garantir la cohérence des données qu'il stocke, même en cas de panne (arrêt brutal) du système,
- assurer les relations entre les données définies par les utilisateurs.

Ce produit est complètement intégré à Windows et ce à plusieurs niveaux :

- Observateur des événements : le journal des applications est utilisé pour consigner les erreurs générées par SQL Server. La gestion des erreurs est centralisée par Windows, ce qui facilite le diagnostic.
- Analyseur de performances : par l'ajout de nouveaux compteurs, il est facile de détecter les goulots d'étranglement et de mieux réagir, pour éviter ces problèmes. On utilise toute la puissance de l'analyseur de performances, et il est possible au sein du même outil de poser des compteurs sur SQL Server et sur Windows et ainsi d'être à même de détecter le vrai problème.
- Traitements parallèles : SQL Server est capable de tirer profit des architectures mutiprocesseurs. Chaque instance SQL Server dispose de son propre processus d'exécution et des threads Windows ou bien des fibres (si l'option est activée) sont exécutés afin d'exploiter au mieux l'architecture matérielle disponible. Chaque instance SQL Server exécute toujours plusieurs threads Windows. Pour prendre en charge tous les processeurs présents sur le système, le paramètre de configuration **max degree of parallelism** doit conserver la valeur 0. Il s'agit de la valeur par défaut. Pour empêcher la génération de plan d'exécution parallèle, il suffit d'affecter la valeur 1 à ce paramètre. Enfin en lui affectant une valeur comprise entre 1 et le nombre de processeurs, il est possible de limiter le degré de parallélisme.
- Sécurité : SQL Server est capable de s'appuyer intégralement sur la sécurité gérée par Windows, afin de permettre aux utilisateurs finaux de ne posséder qu'un nom d'utilisateur et un seul mot de passe. Néanmoins SQL Server gère son propre système de sécurité pour tous les clients non Microsoft.
- Les services Windows sont mis à contribution pour exécuter les composants logiciels correspondant au serveur. La gestion du serveur (arrêt, démarrage et suspension) est facilitée et il est possible de profiter de toutes les fonctionnalités associées aux services de Windows (démarrage automatique, exécution dans le contexte d'un compte d'utilisateur du domaine...).
- Active Directory : les serveurs SQL 2008 et leurs propriétés sont automatiquement enregistrés dans le service

d'annuaire Active Directory. Il est ainsi possible d'effectuer des recherches dans Active Directory pour localiser les instances SQL Server qui fonctionnent.

SQL Server peut gérer deux types de bases de données différentes :

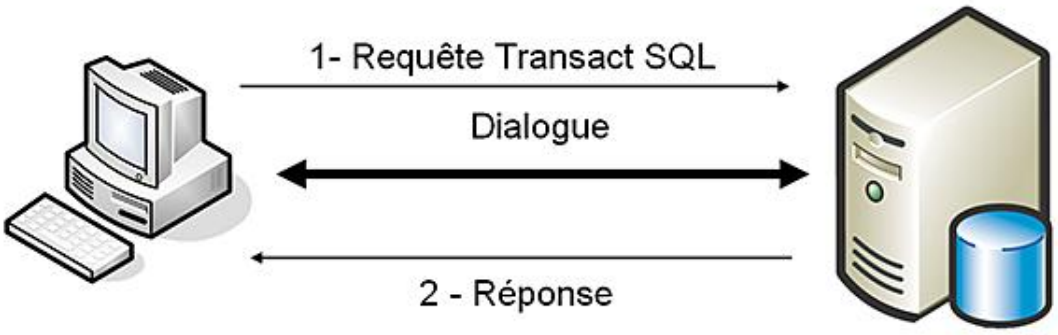
- les bases OLTP (*OnLine Transactional Processing*) qui correspondent à des bases dans lesquelles les informations sont stockées de façon directe afin de réutiliser plus tard ces informations telles qu'elles ont été stockées.

- les bases OLAP (*OnLine Analytical Processing*) qui contiennent des informations statistiques afin d'être capable d'extraire les informations sous forme de cube multidimensionnel dans un but d'aide à la décision par exemple. SQL Express n'étant pas en mesure de gérer des bases de données d'une taille supérieure à 4 Go et n'intégrant pas les outils décisionnels de SQL Server tels que SSAS (*SQL Server Analysis Services*), il n'est donc pas possible de gérer des bases OLAP avec cette édition.

  Les statistiques contenues dans des bases OLAP s'appuient sur des informations contenues dans une base OLTP.

## 2. Mode de fonctionnement Client/Serveur

Toutes les applications qui utilisent SQL Server pour gérer les données, s'appuient sur une architecture client/serveur. L'application cliente est chargée de la mise en place de l'interface utilisateur. Cette application s'exécute généralement sur plusieurs postes clients simultanément. Le serveur, quant à lui, est chargé de la gestion des données, et répartit les ressources du serveur entre les différentes demandes (requêtes) des clients. Les règles de gestion de l'entreprise se répartissent entre le client et le serveur.



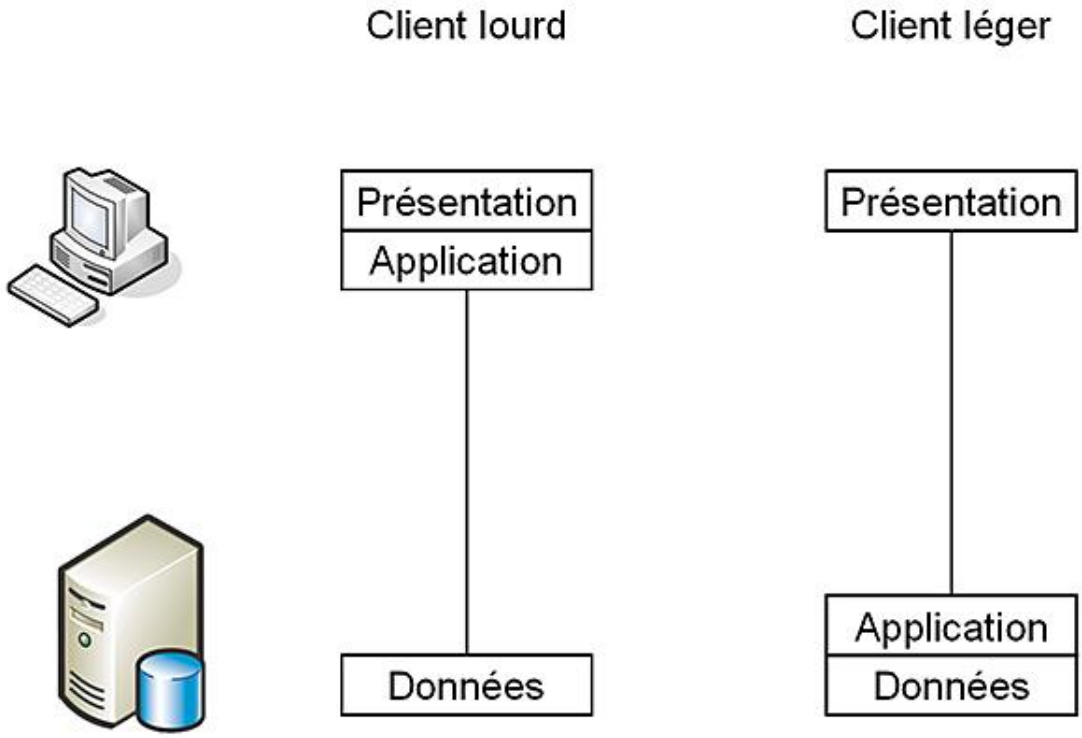

*Mode de fonctionnement Client/Serveur*

On peut distinguer trois cas :

- les règles sont entièrement implémentées sur le client, appelé alors **client lourd**. Cette solution permet de libérer des ressources au niveau du serveur, mais les problèmes de mise à jour des clients et de développement d'autres applications se posent.

- les règles sont entièrement définies sur le serveur, le client est alors un **client léger**. Cette solution permet d'obtenir des clients qui possèdent peu de ressources matérielles, et autorise une centralisation des règles ce qui rend plus souples les mises à jour. Cependant de nombreuses ressources sont consommées sur le serveur et l'interaction avec l'utilisateur risque d'être faible, puisque l'ensemble des contraintes est vérifié lorsque l'utilisateur soumet sa demande (requête) au serveur.

- les règles d'entreprises sont définies sur une tierce machine, appelée **Middle Ware**, afin de soulager les ressources du client et du serveur, tout en conservant la centralisation des règles.




L'architecture client/serveur permet un déploiement optimum des applications clientes sur de nombreux postes tout en conservant une gestion centralisée des données (le serveur), ce qui rend possible le partage d'informations à l'intérieur de l'entreprise.

Il est bien sûr possible d'avoir plusieurs applications clientes sur le même serveur de base de données. Cette possibilité offre de nombreuses fonctionnalités, mais il faut toutefois veiller à ce que la charge de travail sur le serveur ne soit pas trop importante au regard des capacités de la machine.

Cette architecture client/serveur est respectée par tous les outils permettant d'accéder à des informations contenues par le serveur SQL, donc les outils d'administration, même s'ils sont installés sur le serveur.

Toutes les demandes en provenance des clients vers le serveur, doivent être écrites en Transact-SQL. Ce langage de requête de base de données respecte la norme ANSI SQL-92. Le SQL fournit un ensemble de commandes pour gérer les objets et manipuler les données dans les bases. Le Transact SQL est enrichi de nombreuses fonctionnalités, non normalisées, afin d'étendre les possibilités du serveur. Il est ainsi possible de définir des procédures stockées sur le serveur.

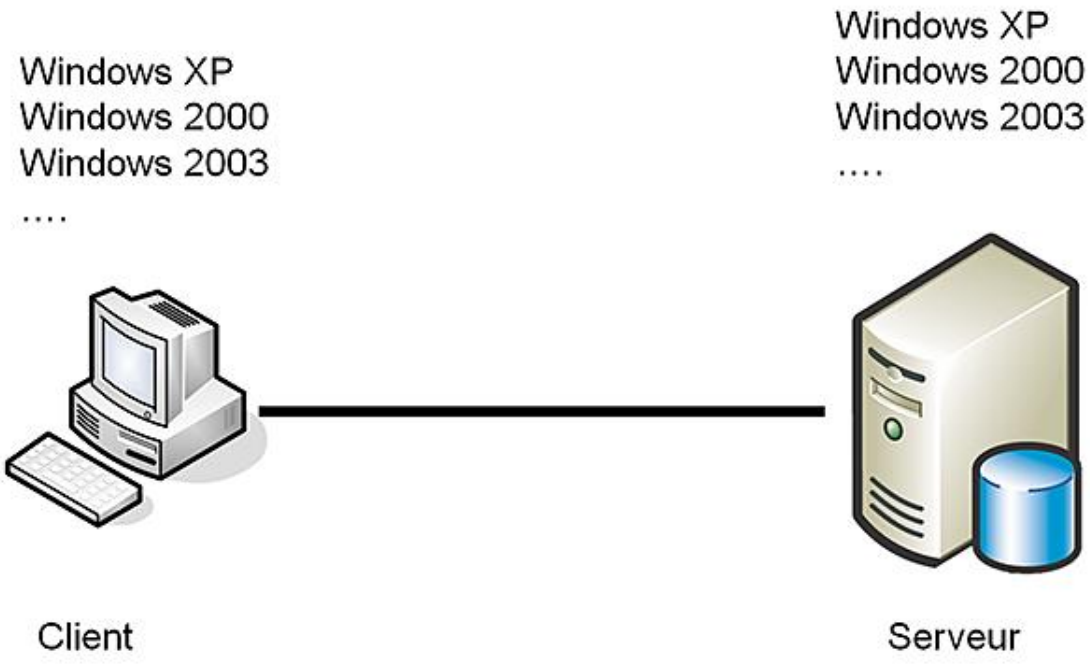
## 3. Les plates-formes possibles

Il est important de distinguer deux cas : d'un côté les plates-formes possibles pour le client et de l'autre les plates-formes pour le serveur.

Les plates-formes clientes présentées ici sont les postes sur lesquels les outils d'administration SQL Server peuvent être installés. Il ne s'agit pas des postes qui hébergent une application qui se connecte à une instance SQL Server pour gérer les données.

D'une façon synthétique, les outils clients d'administrations peuvent être installés sur tous systèmes d'exploitation Windows 2003, Windows XP Pro ou tout système plus récent.

Par contre, pour la partie serveur, les disponibilités en termes de plates-formes sont fonctions de l'édition SQL Server choisie. Néanmoins pour héberger une instance de base de données en production, il est nécessaire de disposer d'un serveur performant et fiable. Une plate-forme Windows 2003 ou 2008 est donc recommandée. L'édition de Windows 2003 ou 2008 sera choisie en fonction des contraintes imposées par l'édition SQL Server sélectionnée et des contraintes liées à l'environnement technique. L'installation d'une instance sous Windows XP, Vista, 7 sera réservée à des postes nomades.




Dans le cas où le client héberge une application spécifique, la gamme des plates-formes est considérablement élargie grâce, en particulier, au pilote jdbc qui permet d'accéder à une instance SQL Server depuis une application écrite en java. La gamme est encore élargie dans les cas d'une application ASPX qui propose une interface Internet. Un simple navigateur Internet permet alors de lancer l'application.

## 4. Les composants de SQL Server

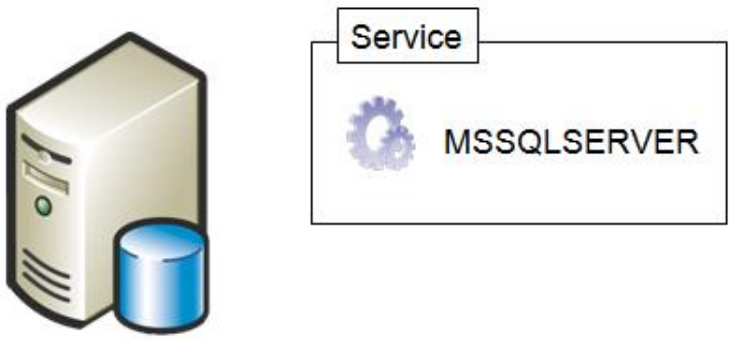
Le moteur de base de données de SQL Server ou Database Engine est composé de plusieurs logiciels. Certains s'exécutent sous forme de services alors que d'autres possèdent une interface utilisateur graphique ou en ligne de commande.

### Composants Serveur

SQL Server s'exécute sous forme de services Windows. Suivant les options d'installation choisies, il peut y avoir plus de services. Les principaux services sont :

- SQL Server : c'est le serveur de base de données à proprement parlé. Si ce service n'est pas démarré, il n'est pas possible d'accéder aux informations. C'est par l'intermédiaire de ce service que SQL Server assure la gestion des requêtes utilisateurs. Ce service est référencé sous le nom MSSQLSERVER pour l'instance par défaut et MSSQLSERVER $*nomInstance* dans le cas d'une instance nommée.

- Microsoft Full Text Search : ce service propose de gérer l'indexation des documents de type texte stockés dans SQL Server et gère également les recherches par rapport aux mots clés. Cette fonctionnalité n'est disponible que dans l'édition SQL Server 2008 Express Advanced.




Il est possible d'installer plusieurs instances de SQL Server sur le même poste.

### Connectivité Client

L'installation des composants de connectivité sur les postes clients permet de prendre en charge la gestion du réseau, la bibliothèque SQL Native pour les programmes en accès natif, le support OLE-DB et ODBC.

### Outils de gestion

Les réalisations des tâches d'administration sont possibles par l'utilisation d'outils. Ces outils possèdent pour la plupart une interface graphique conviviale et d'utilisation intuitive. Cependant, les tâches administratives doivent être réfléchies avant leur réalisation. L'utilisation de certains outils suppose que le composant serveur correspondant est installé.

Ces outils sont :

- SQL Server Management Studio pour réaliser toutes les opérations au niveau du serveur de base de données. Seule l'édition Express Advanced permet d'installer une version Express de SQL Server Management Studio. Toutefois, lorsque SQL Server Express est utilisé dans le cadre de la réalisation d'une application, toutes les opérations sont bien souvent faites directement au travers de Visual Studio.

- SQL Server Configuration Manager pour gérer les services liés à SQL Server.

Tous les outils et le fonctionnement de SQL Server sont richement documentés dans la documentation en ligne.

### Les composants

Les différentes briques logicielles fournies par SQL Server s'articulent toujours autour du moteur de base de données relationnelles qui traite de façon performante les informations stockées au format relationnel et au format xml.

- SQL Server Integration Service (SSIS) est un outil d'importation et d'exportation de données facile à mettre en place tout en étant fortement paramétrable. SQL Server Express ne bénéficie pas de l'ensemble des fonctionnalités de SSIS, il n'est en effet pas possible de définir des lots complexes. Seules les définitions d'opérations simples d'importation ou exportation de données sont possibles. Il est par contre tout à fait possible d'exécuter un lot défini à l'aide d'une autre édition de SQL Server.

- Reporting Services permet de mettre en place des rapports d'analyse des données.

- La réplication des données sur différentes instances permet de positionner les données au plus près des utilisateurs et de réduire les temps de traitement.

- L'intégration du CLR dans SQL Server permet de développer procédures et fonctions en utilisant les langages VB.Net et C#. L'intégration du CLR ne vient pas se substituer au Transact SQL mais se présente comme un complément afin de pouvoir réaliser un codage simple et performant pour l'ensemble des fonctionnalités qui doivent être présentes sur le serveur.

### Mémoire AWE

Une meilleure gestion de la mémoire est proposée avec la mise en place de l'API AWE qui permet de gérer, sur des systèmes 32 bits, plus de 4 Go de mémoire. L'édition entreprise est ainsi capable de gérer jusqu'à 64 Go de mémoire. La prise en charge de AWE est possible en activant l'option de configuration awe enabled avec sp_configure.

awe enabled est une option de configuration avancée.

### Reporting Services

Reporting Services permet la création de rapports pour présenter au mieux les informations contenues dans SQL Server.

Les fonctionnalités de rapports ne sont disponibles que dans l'édition Express Advanced de SQL Server 2008. De plus, le moteur de rapport possède des limites bien précises comme le fait de ne pas pouvoir exploiter plus de 1 Go de mémoire, ou bien l'impossibilité de s'intégrer avec SharePoint.

### CLR

L'intégration du CLR (*Common Language Runtime*) à SQL Server, permet d'augmenter considérablement les possibilités offertes en terme de programmation. La présence du CLR ne remet pas en cause le Transact SQL. Chacun est complémentaire. Le Transact SQL est parfait pour écrire des procédures ou fonctions pour lesquelles il y a un

traitement intensif des données. Au contraire, dans le cas où le volume des données manipulées est faible, le CLR permet d'écrire simplement des traitements complexes, car il bénéficie de toute la richesse du CLR.

Le CLR permet également de définir ses propres types de données ou bien de nouvelles fonctions de calcul d'agrégat.

Enfin, le CLR permet aux développeurs d'applications de développer des procédures et fonctions sur SQL Server tout en conservant leurs langages favoris (VB.Net ou C# par exemple), et donc sans avoir besoin de maîtriser le Transact SQL.

Dans le cas où le code est écrit depuis Visual Studio, l'intégration de la version compilée dans SQL Server et le mappage CLR-Transact SQL est réalisé de façon automatique. Il est possible de réaliser le développement en dehors du Visual Studio mais l'intégration à SQL Server sera faite de façon manuelle, ce qui est une tâche fastidieuse.

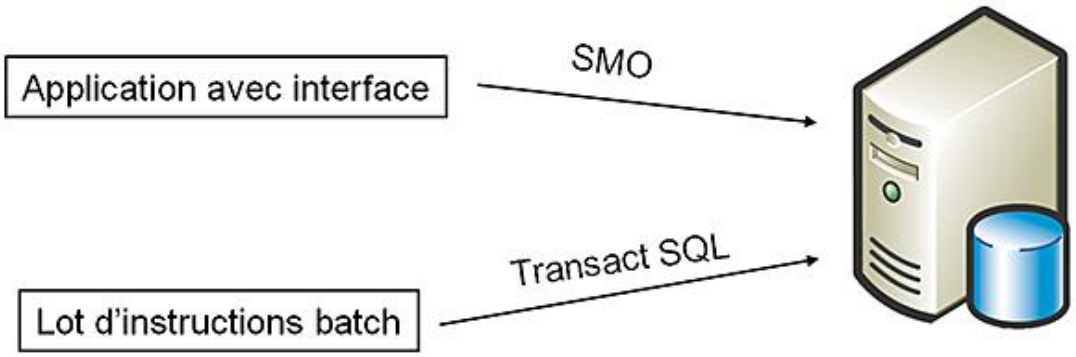
# Architecture

## 1. Administration

Le langage naturel de SQL Server est le Transact SQL. Il est donc nécessaire de lui transmettre les instructions dans ce langage. Comme ce langage n'est pas forcément naturel pour l'utilisateur, il est possible de composer l'instruction de façon graphique par SQL Server Management Studio, puis de provoquer son exécution sur le serveur à l'aide des boutons **OK, Appliquer**... Les outils graphiques utilisent la bibliothèque SMO (*SQL Server Management Object*) pour établir un dialogue efficace avec le serveur.

SQL SMO englobe et étend SQL DMO. La bibliothèque SMO est donc compatible avec SQL Server 7, SQL Server 2000, 2005 et 2008.

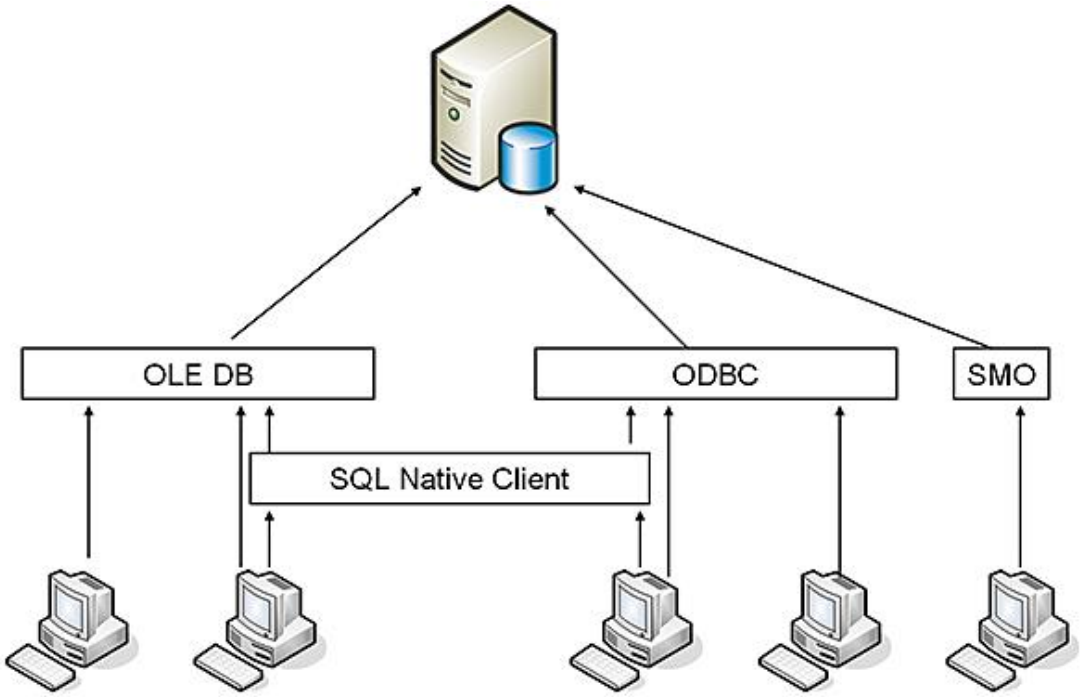
Il est donc possible d'écrire des scripts Transact SQL pour exécuter des opérations administratives sous forme de traitement batch.




*Administration de SQL Server*

## 2. Programmation

Le développement d'applications clientes pour visualiser les données contenues dans le serveur peut s'appuyer sur différentes technologies.




La DLL SQL Native Client est une méthode d'accès aux données qui est disponible aussi bien en utilisant la

technologie OLE-DB ou bien ODBC pour accéder aux données. Avec cette nouvelle API, il est possible d'utiliser l'ensemble des fonctionnalités de SQL Server comme les types personnalisés définis avec les CLR (UDT : *User Defined Type*), MARS ou bien encore le type xml.

SQL Native Client est une API qui permet de tirer pleinement profit des fonctionnalités de SQL Server et de posséder un programme qui accède de façon optimum au serveur.

Il est toujours possible d'utiliser les objets ADO pour accéder à l'information. Ce choix est plus standard car un programme accédant à une source de données ADO peut travailler aussi bien avec une base Oracle que SQL Server, mais ne permet pas la même gestion de toutes les fonctionnalités offertes par SQL Server. L'API SQL Native Client permet l'écriture de programmes clients optimisés mais uniquement capables d'accéder à des données hébergées par un serveur SQL Server.

SQL Native Client sera adopté comme modèle d'accès aux données dans les nouveaux programmes écrits en VB.Net ou C# qui souhaitent travailler avec SQL Server mais aussi dans les programmes existants lorsque ces derniers souhaitent travailler avec des éléments spécifiques à SQL Server, comme le type XML, par exemple.

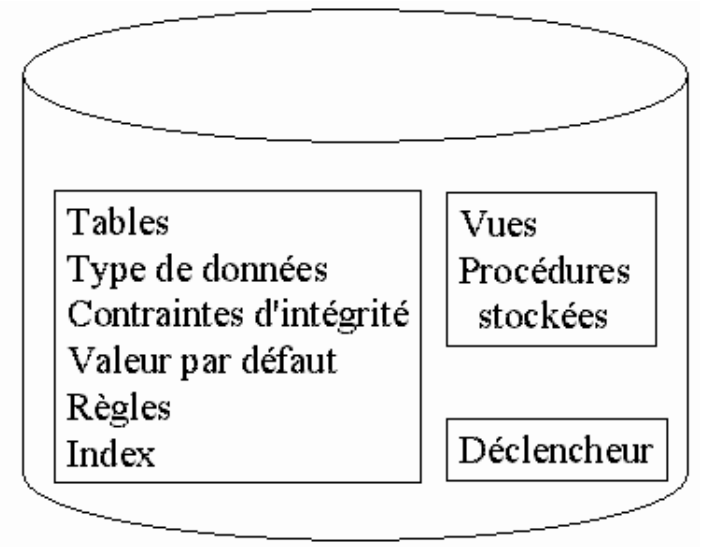
Ce modèle de programmation correspond à une application client qui souhaite gérer les données. Dans le cas où l'application souhaite être capable de faire des opérations d'administration, il est alors nécessaire d'utiliser la bibliothèque SMO.

# Base de données SQL Server

## 1. Objets de base de données

Les bases de données contiennent un certain nombre d'objets logiques. Il est possible de regrouper ces objets en trois grandes catégories :

- Gestion et stockage des données : tables, type de données, contraintes d'intégrité, valeur par défaut, règles et index.
- Accès aux données : vues et procédures stockées.
- Gestion de l'intégrité complexe : déclencheur (procédure stockée s'exécutant automatiquement lors de l'exécution d'un ordre SQL modifiant le contenu d'une table : INSERT, UPDATE et DELETE). Le déclencheur est toujours associé à une table et à une instruction SQL. Il permet de mettre en place des règles d'intégrité complexes à cheval sur plusieurs tables ou de maintenir des données non normalisées.



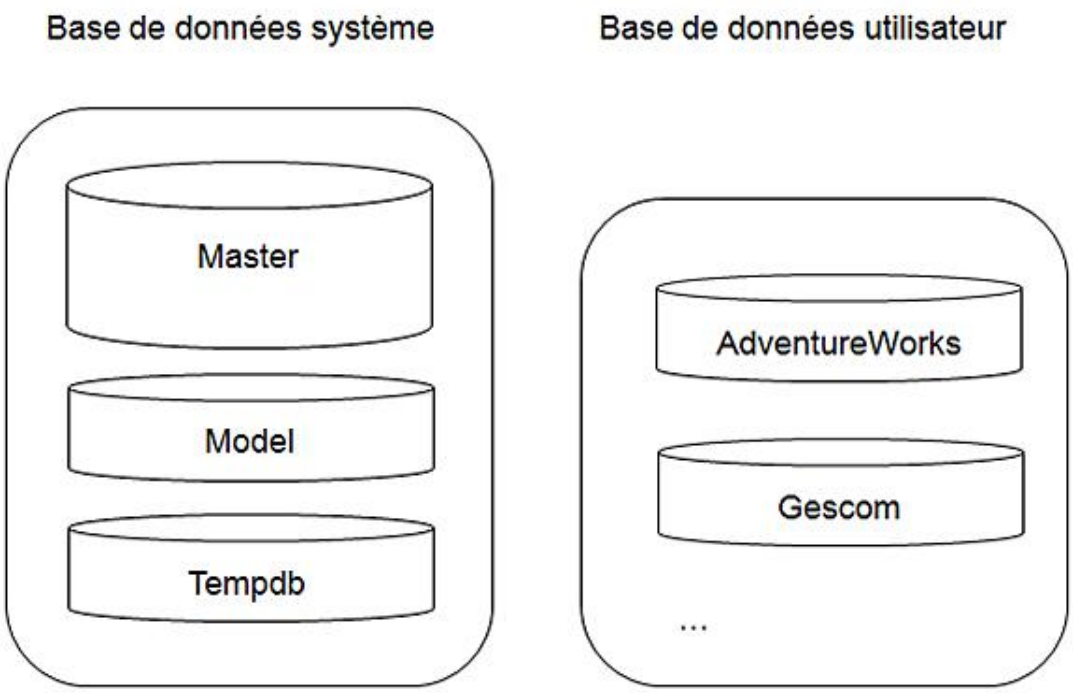

*Objet de base de données*

Nom complet des **objets**

La règle appliquée pour nommer les objets permet une parfaite identification. Le nom complet est composé comme suit : *serveur.nomBase.propriétaire.objet*. Par défaut, seul le nom de l'objet est précisé. Cette notion sera détaillée au cours du chapitre Gestion de la base de données.

## 2. Bases de données système et tables système

Pour gérer l'ensemble des données stockées, SQL Server s'utilise lui-même. Il existe donc des bases de données système et sur chaque base utilisateur, quelques tables système. L'insertion et la mise à jour de données dans ces tables ne s'effectuent jamais directement, mais via des commandes Transact SQL ou des procédures stockées.




*Organigramme des bases de données*

Les noms des bases de données et des tables système sont fixés et connus par SQL Server. Il ne faut donc pas renommer une table ou une base système.

**Master**

C'est la base de données principale de SQL Server. L'ensemble des données stratégiques pour le bon fonctionnement du serveur y est stocké (comptes de connexion, options de configuration, l'existence des bases de données utilisateurs et les références vers les fichiers qui composent ces bases...).

**Model**

Cette base contient l'ensemble des éléments inscrits dans toute nouvelle base utilisateur. Par défaut, il n'y a que les tables système, mais il est possible de rajouter des éléments.

**Tempdb**

La base **Tempdb** est un espace temporaire de stockage partagé. Il permet de gérer les tables temporaires locales ou globales, les tables de travail intermédiaires pour faire des tris par exemple, mais aussi les jeux de résultats des curseurs. La base **Tempdb** est recréée, avec sa taille initiale, lors de chaque démarrage de l'instance. Ainsi, aucune information ne peut être conservée de façon persistante à l'intérieur de la base **Tempdb**. Les objets temporaires sont, quant à eux, supprimés lors de la déconnexion de leur propriétaire.

**Bases de données utilisateur**

Les bases de données utilisateurs vont héberger les données fournies par les utilisateurs. Les bases présentes sur le schéma précédent (AdventureWorks et Gescom) sont les bases d'exemples utilisées dans la documentation officielle de SQL Server et dans cet ouvrage.

## 3. Les tables système

Les tables système sont toujours présentes dans SQL Server. Cependant, il est recommandé de ne pas travailler directement avec ces tables. Pour rechercher l'information, il faut passer par le schéma d'information et plus exactement les vues définies sous le schéma de l'utilisateur **sys** lorsque cela est possible.

Dans le tableau ci-dessous, quelques tables système sont référencées.

| **Catalogue système (présent uniquement dans la base Master)** |
|---|

| Table système | Fonction |
|---|---|
| syslogins | Une ligne pour chaque utilisateur ou groupe Windows autorisé à se connecter au serveur SQL. |
| sysmessages | Une ligne pour chaque message d'erreur défini et pour chaque langue. |
| sysdatabases | Une ligne par base de données utilisateur. |
| sysconfigures | Une ligne pour chaque option de configuration du serveur. |
| sysusers | Une ligne pour chaque utilisateur défini dans la base. |
| syscolumns | Une ligne pour chaque colonne des tables, vues et pour chaque paramètre des procédures stockées. |
| sysobjects | Une ligne pour chaque objet de la base de données. |

Les tables système sont utilisées directement par le moteur de SQL Server. Les applications qui utilisent SQL Server ne doivent en aucun cas accéder directement à ces tables, même en lecture. En effet, comme la structure de ces tables évolue avec les versions de SQL Server, si certaines applications accèdent de façon directe aux tables système, on peut se trouver dans l'impossibilité de migrer vers une nouvelle version de SQL Server tant que l'application n'a pas été réécrite.

SQL Server ne prend pas en compte les déclencheurs qui pourraient être définis sur les tables système car ils peuvent gêner le bon déroulement de certaines opérations.

## 4. Extraction de méta-données

Pour interroger les données contenues dans les tables système, il est déconseillé de le faire directement par une requête de type SELECT. Il est préférable de passer par l'utilisation de procédures stockées, de fonctions système et de vues du schéma d'information.

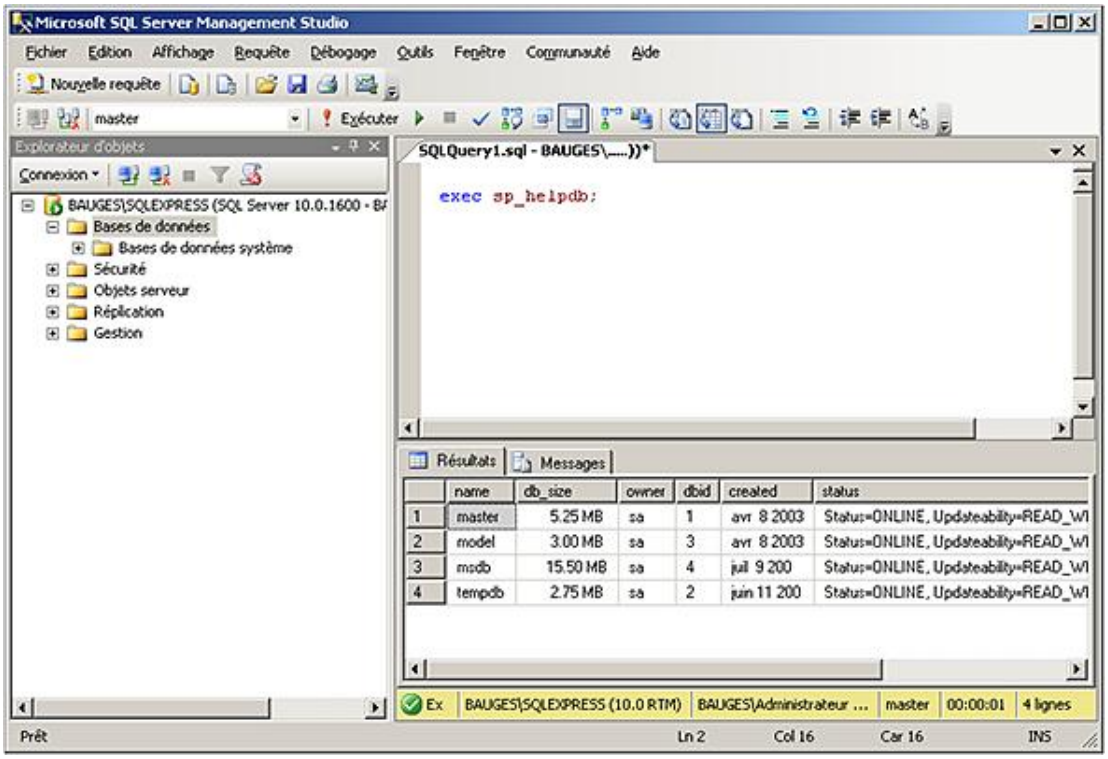
En tant qu'administrateur, il est possible de modifier le contenu des tables système. Cette opération est à proscrire car elle peut avoir des conséquences irréversibles et dramatiques. Le seul moyen de remédier à un tel problème sera alors de restaurer une sauvegarde.

### Procédures stockées système

Les procédures stockées système sont maintenues, pour la plupart, pour des raisons de compatibilité ascendante. Leur utilisation est donc à déconseiller.

Pour interroger les tables système, il existe de nombreuses procédures stockées. Elles commencent toutes par **sp_**. Parmi toutes les procédures stockées, citons :

| Procédure stockée | Description |
|---|---|
| sp_help [nom_objet] | Informations sur l'objet indiqué. |
| sp_helpdb [nom_base_données] | Informations sur la base de données indiquée. |
| sp_helpindex [nom_table] | Informations sur les index de la table indiquée. |
| sp_helplogins [nom_connexion] | Informations sur la connexion indiquée. |
| sp_who | Liste des utilisateurs actuellement connectés. |




## Le catalogue

SQL Server propose des vues système qui permettent d'obtenir des informations système. Toutes ces vues sont présentes dans le schéma **sys**.

Afin de naviguer au mieux dans ces vues, elles sont regroupées par thèmes :

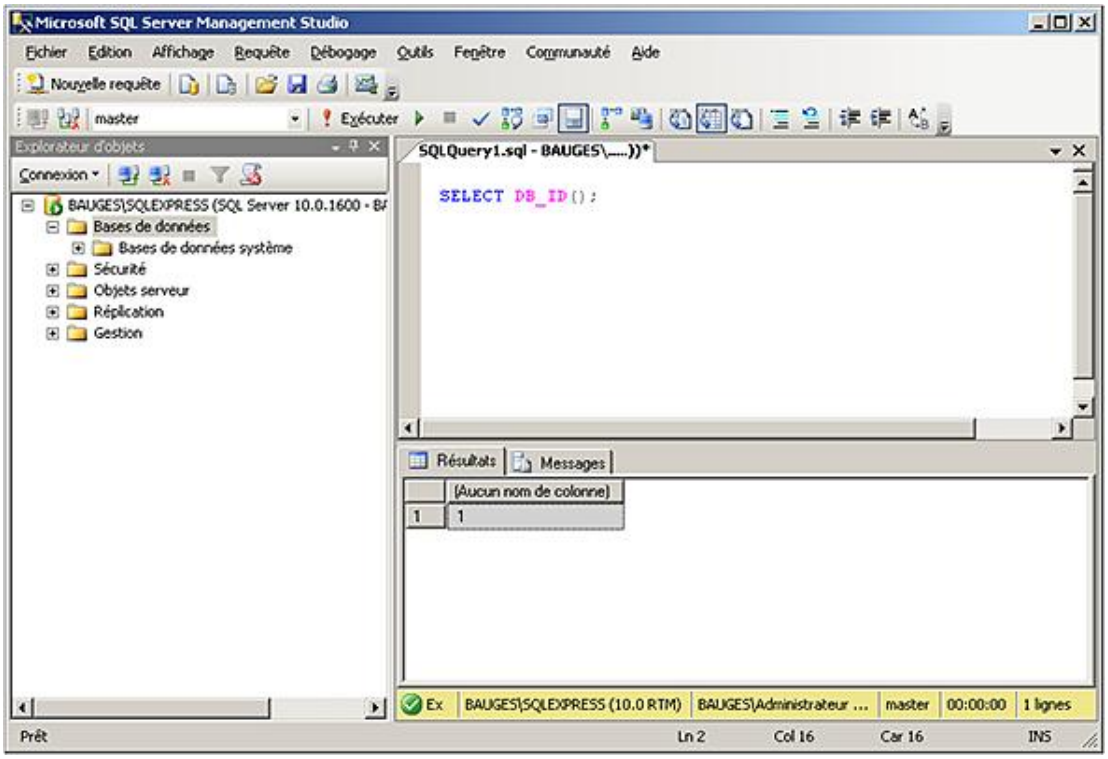
- Objets, types et index
- Serveurs liés
- CLR
- Mise en miroir
- Service Broker
- Sécurité
- Transactions
- Configuration du serveur
- Information sur le serveur
- Environnement d'exécution
- Stockage
- Point de terminaisons
- Partitionnement
- Traces et évènements

## Fonctions système

Les fonctions système sont utilisables avec des commandes Transact SQL. Il est ainsi possible de récupérer des valeurs concernant la base de données sur laquelle on travaille, sur le serveur ou sur les utilisateurs.

Citons-en quelques-unes :

| Fonctions système | Paramètre | Description |
|---|---|---|
| DB_ID | Nom | Retrouve l'identificateur de la base de données. |
| USER_NAME | ID | Retrouve le nom de l'utilisateur à partir de son identifiant. |
| COL_LENGTH | Colonne | Longueur de la colonne. |
| STATS_DATE | Index | Date de dernière mise à jour des statistiques. |
| DATALENGTH | Type de données | Longueur de l'expression. |




## Schéma d'information

Il s'agit d'un ensemble de vues qui proposent une visualisation des paramètres de façon indépendante des tables système. En ne faisant pas directement référence aux tables système, on se préserve des modifications qui peuvent intervenir sur leurs structures lors des prochaines versions. De plus, ces vues sont conformes à la définition du standard ANSI SQL pour les schémas d'information. Chaque vue présente des méta-données pour l'ensemble des objets contenus dans la base de données.

Les vues du schéma d'information :

CHECK_CONSTRAINTS

Visualise l'ensemble des contraintes de type CHECK définies dans la base de données.

COLUMN_DOMAIN_USAGE

Visualise l'ensemble des colonnes définies sur un type de données utilisateur.

COLUMN_PRIVILEGE

Visualise l'ensemble des privilèges accordés, au niveau colonne, pour l'utilisateur courant ou pour un utilisateur de la base de données.

COLUMNS

Visualise la définition de l'ensemble des colonnes accessibles, dans la base de données courante, à l'utilisateur actuel.

CONSTRAINT_COLUMN_USAGE

Visualise l'ensemble des colonnes pour lesquelles il existe une contrainte.

CONSTRAINT_TABLE_USAGE

Visualise l'ensemble des tables pour lesquelles au moins une contrainte est définie.

DOMAIN_CONSTRAINTS

Visualise l'ensemble des types de données utilisateurs, qui sont accessibles par l'utilisateur courant et qui sont liés à une règle.

DOMAINS

Visualise l'ensemble des types de données utilisateurs, qui sont accessibles par l'utilisateur courant.

KEY_COLUMN_USAGE

Visualise l'ensemble des colonnes pour lesquelles une contrainte de clé est définie.

PARAMETERS

Visualise les propriétés des paramètres des fonctions définies par l'utilisateur et des procédures stockées accessibles à l'utilisateur courant. Pour les fonctions, les informations relatives à la valeur de retour sont également visualisées.

REFERENTIAL_CONSTRAINTS

Visualise l'ensemble des contraintes de clés étrangères définies dans la base de données courante.

ROUTINES

Visualise l'ensemble des procédures stockées et des fonctions accessibles à l'utilisateur courant.

ROUTINE_COLUMNS

Visualise les propriétés de chaque colonne renvoyées par les fonctions accessibles à l'utilisateur courant.

SCHEMATA

Visualise les bases de données pour lesquelles l'utilisateur courant possède des autorisations.

TABLE_CONSTRAINTS

Visualise les contraintes de niveau table, posées dans la base de données courante.

TABLE_PRIVILEGES

Visualise l'ensemble des privilèges accordés à l'utilisateur actuel dans la base de données courante et l'ensemble des privilèges accordés par l'utilisateur actuel à d'autres utilisateurs de la base de données courante.

TABLES

Visualise l'ensemble des tables de la base de données courante pour lesquelles l'utilisateur actuel dispose de droits.

VIEW_COLUMN_USAGE

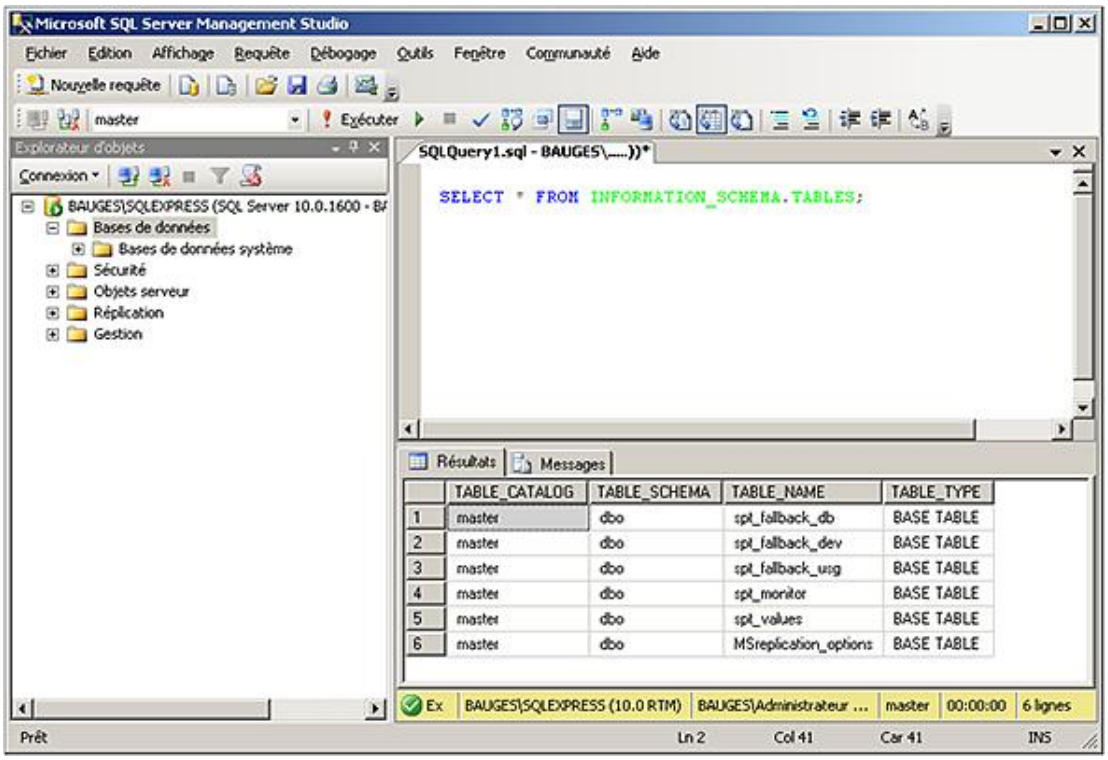
Visualise l'ensemble des colonnes de tables qui sont utilisées dans la définition d'une ou plusieurs vues.

VIEW_TABLE_USAGE

Visualise l'ensemble des tables de la base de données courante qui sont utilisées lors de la définition de vues.

VIEWS

Visualise l'ensemble des vues accessibles à l'utilisateur actuel dans la base de données courante.




## 5. Les tâches de l'administrateur

L'administrateur de bases de données a pour objectif principal d'améliorer le fonctionnement de la base de données. Bien que SQL Server possède de nombreux outils et algorithmes d'auto-optimisation, il reste de nombreuses tâches à l'administrateur.

Les principales tâches de l'administrateur sont :

- Gérer les services SQL Server ;
- Gérer les instances SQL Server ;
- Mettre en place le processus de sauvegarde et de restauration ;
- Configurer une disponibilité des données en accord avec la politique d'entreprise ;
- Gérer les configurations réseaux ;
- Importer et exporter des données.

En plus des compétences système que doit posséder l'administrateur pour être capable de gérer au mieux l'instance

SQL Server, il est important pour lui de connaître les possibilités offertes par SQL Server pour l'automatisation des tâches avec SQL Agent.

Pour mesurer le résultat de son travail et comparer les différents choix de configuration qu'il peut être amené à faire, l'administrateur doit être en mesure d'utiliser les outils et méthodes liés à SQL Server.

Enfin et c'est sans doute un point crucial, l'administration de la base doit être inscrite dans un processus plus global qui intègre l'administrateur dès la conception de la base, de façon à effectuer les meilleurs choix en terme d'architecture dès la conception. L'administrateur pourra ainsi intervenir sur la création de la base et les choix faits comme, par exemple, le nombre de groupe de fichiers à utiliser, les index, les vues, les procédures stockées à définir de façon à optimiser le trafic réseau mais aussi pour simplifier la gestion des droits d'accès. C'est également l'administrateur qui va pouvoir conseiller sur le partitionnement ou non d'une table.

# Installer SQL Server

L'installation de SQL Server permet de présenter les différentes éditions de SQL Server ainsi que de détailler les options d'installation possibles. Un point tout particulier sera mis en place pour détailler l'exécution des services associés à SQL Server (logiciels serveur). Une fois le serveur installé, il faut s'assurer que l'installation s'est déroulée correctement puis il faut configurer le serveur et certains outils clients pour réaliser les différentes étapes d'administration.

Bien que facile à réaliser, l'installation de SQL Server doit être une opération réfléchie. En effet, de nombreuses options d'installation existent et leur choix doit correspondre à un réel besoin, ou permettre de couvrir une évolution future du système.

## 1. Les éditions de SQL Server

SQL Server est disponible sous la forme de plusieurs éditions. Chaque édition se distingue par des caractéristiques spécifiques. En fonction des possibilités souhaitées, le choix se portera sur une édition ou bien une autre.

Certaines éditions (Standard et Entreprise) sont plus destinées à la gestion des données d'une entreprise tandis que certaines éditions (Developer, Workgroup, Web, Express) ont pour objectif de satisfaire des besoins bien particuliers. Toutes ces éditions de SQL Server sont disponibles pour des plates-formes 32 et 64 bits. Enfin, l'édition Compact est quant à elle destinée aux périphériques mobiles.

L'édition Express de SQL Server 2008 possède la particularité d'être utilisable en production sans qu'il soit nécessaire de s'acquitter d'une licence de SQL Server. Cette version n'est pas une version dégradée de SQL Server, mais il s'agit bien du moteur SQL Server pleinement fonctionnel. Il n'existe pas de limite quant au nombre d'utilisateurs connectés. Les seules limitations qui existent concernent le volume de données : 4 Go et le fait que le moteur ne puisse pas exploiter plus d'un gigaoctet de mémoire. Il est toutefois raisonnable de penser que lorsqu'une application atteint ces limites, l'entreprise possède les moyens nécessaires pour acquérir une version complète de SQL Server.

Cette édition Express est à conseiller sans modération auprès des développeurs d'applications car il sera possible de migrer très simplement vers une édition supérieure de SQL Server.

Ce type d'édition est également bien adapté pour les applications autonomes. En effet, l'édition Express peut être installée sur une plate-forme Windows utilisateur, comme par exemple l'ordinateur portable d'un commercial qui emmène avec lui la base de tous les articles de l'entreprise. Cette base peut être mise à jour par le processus de réplication de SQL Server qui permet de synchroniser les données au niveau du catalogue et de les injecter dans le Système d'Information de l'entreprise.

Cette édition est également utile dans la cadre d'une application monoposte qui nécessite une gestion robuste et fiable des données. Choisir d'utiliser SQL Server pour ce type d'application permet de laisser ouvert le passage vers une gestion multi-utilisateur.

SQL Server 2008 propose également une édition Express Advanced qui propose les mêmes fonctionnalités que l'édition Express, enrichie de la possibilité d'effectuer des rapports.

Pour compléter cette présentation de l'édition Express de SQL Server, énumérons les principales caractéristiques du produit.

- Sécurité : audit possible et suivi de la conformité C2.
- Prise en charge du code CLR.
- Support complet du type XML et des index associés.
- Prise en charge des types de données spécifiques SQL Server 2008 : date et heure, FILESTREAM, données spatiales.

## 2. Déroulement de l'installation

Comme pour de nombreux produits le processus d'installation se déroule en trois temps bien distincts :

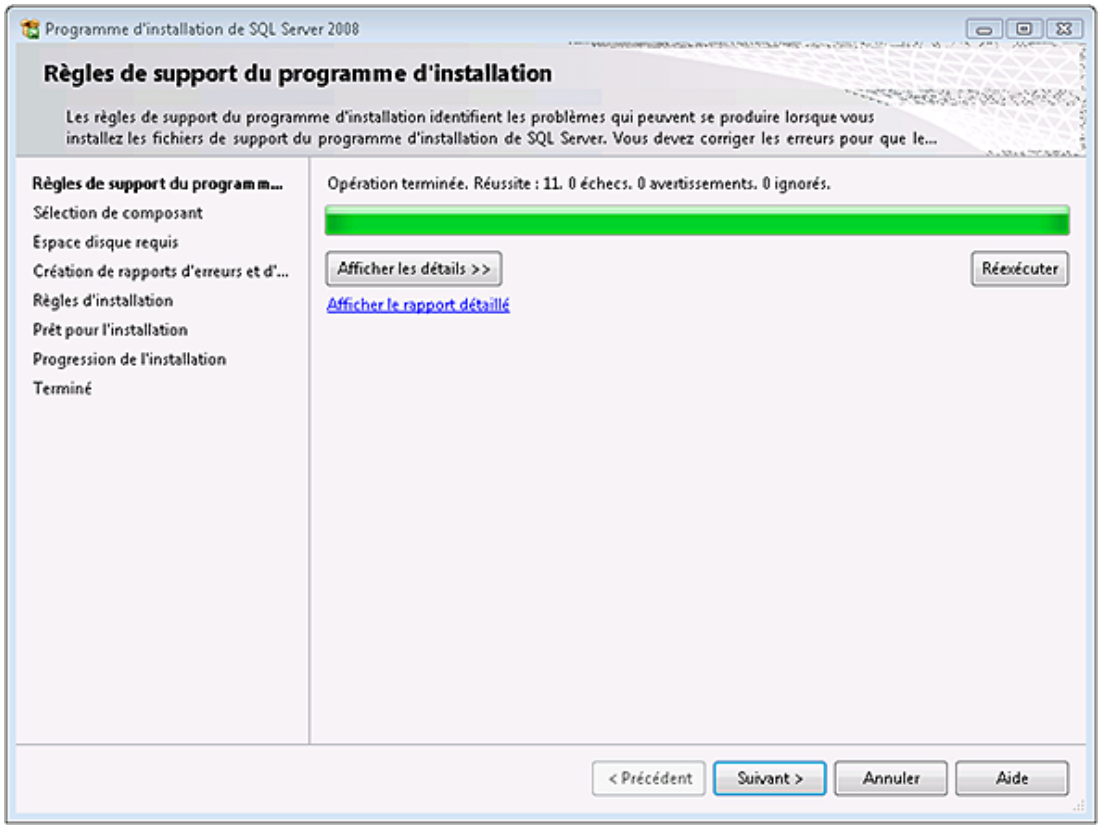
- L'analyse de l'environnement et l'installation de composants nécessaires à la bonne exécution du processus d'installation.
- Le paramétrage des différents composants à installer.

- L'installation des composants sélectionnés au préalable.

Le processus d'installation commence par s'assurer que la version 3.5 du Framework .Net est bien présente sur le système et lance son installation si ce n'est pas le cas.

Par la suite, le premier écran permet de planifier au mieux l'installation de SQL Server 2008 en consultant la documentation relative aux différents cas possibles d'installation/migration de données et en s'assurant que la plate-forme choisie est prête pour une installation de SQL Server 2008. Ce dernier point est assuré par l'outil d'analyse de configuration système. Lors du processus d'installation d'une nouvelle instance de SQL Server 2008, l'outil d'analyse de configuration système est également exécuté. Après saisie de la clé du produit et acceptation du contrat de licence, les fichiers de support du programme d'installation de SQL Server sont installés.

Le processus d'installation de SQL Server 2008 commence par exécuter un certain nombre de règles afin de valider la configuration de la plate-forme.

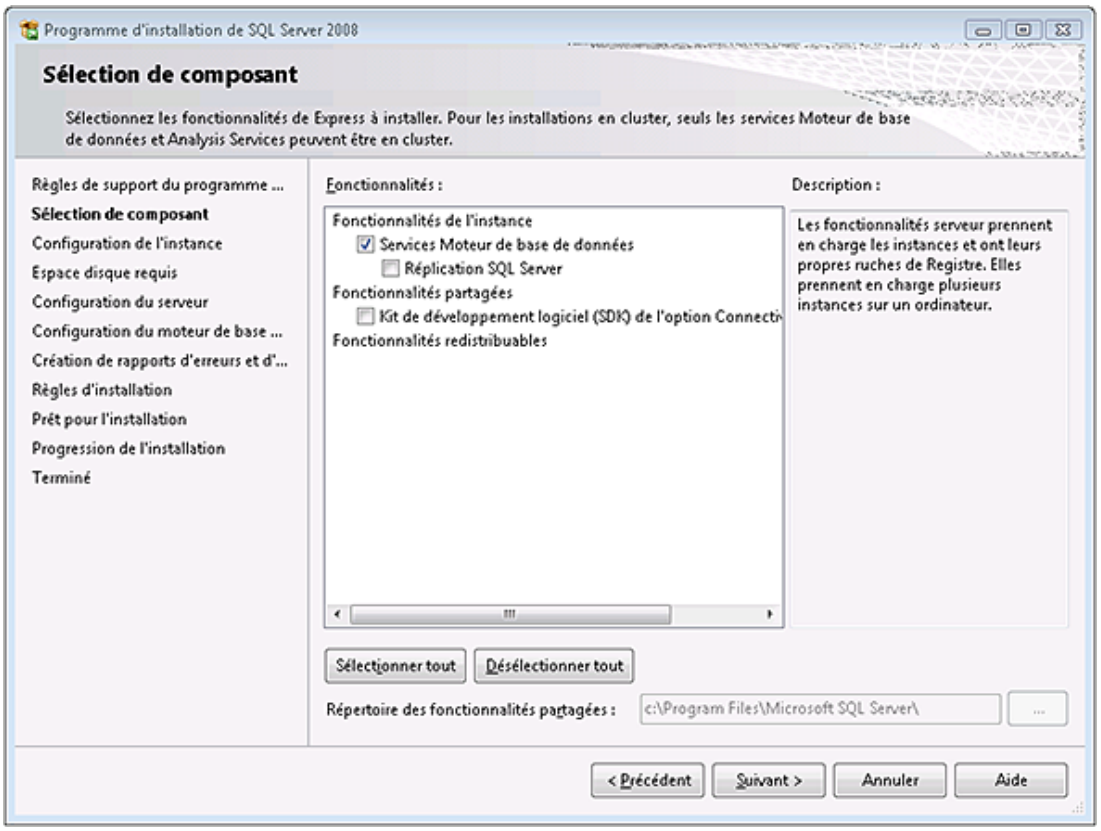



Si ces règles ne sont pas parfaitement respectées il en ressort soit des avertissements, soit des erreurs. Un avertissement permet de préciser que bien qu'il soit possible d'installer une instance SQL Server, certains composants ne pourront pas être installés.

### a. Choix des composants

Avant de procéder au paramétrage de l'installation, SQL Server demande de sélectionner les composants qui vont être installés sur le poste local. C'est à ce niveau qu'il est possible de définir que seuls les outils doivent être installés, ou bien le moteur de base de données.

Il s'agit de sélectionner finement les composants à installer. Il ne s'agit pas de cocher toutes les options, mais bien de sélectionner les composants (client ou serveur) qui vont être effectivement utilisés. En limitant le nombre de composant installés, la surface d'attaque du système est réduite et l'on évite également une surcharge du système par des composants non utilisés. Bien entendu, si certains composants n'ont pas été sélectionné lors de l'installation initiale et que le besoin s'en fait sentir par la suite, il est possible de les ajouter.

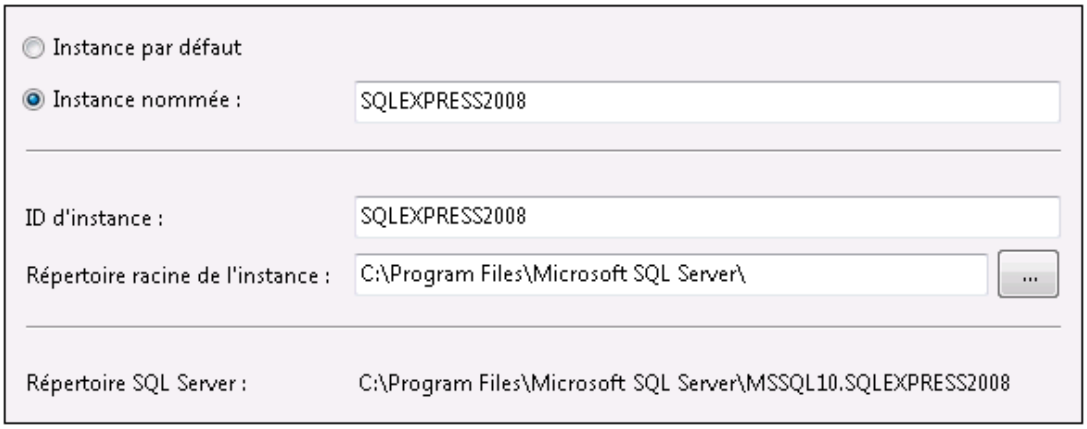



Pour chacun des composants, il est possible de définir son emplacement physique sur le disque dur. Pour accéder à cet écran de configuration, il faut sélectionner le bouton ... .

### b. Nom de l'instance

MS SQL Server offre la possibilité d'installer une ou plusieurs instances du moteur de base de données sur le même serveur. Chaque instance est complètement indépendante des autres installées sur le même serveur et elles sont gérées de façon autonome. Lorsque plusieurs instances sont présentes sur le même serveur, il est nécessaire d'utiliser le nom de chacune d'entre elles pour les distinguer. La première instance installée est généralement l'instance par défaut et elle porte le nom SQL Express.

Avec l'édition Express de SQL Server 2008, il est possible d'installer jusqu'à 16 instances distinctes sur un même serveur.




Après le choix du nom de l'instance à installer, le processus d'installation vérifie les capacités disque du système.

Un même serveur ne peut héberger qu'une seule instance par défaut.

Chaque instance est parfaitement identifiée par son nom. Le nom de l'instance respecte les règles suivantes :

- le nom de l'instance est limité à 16 caractères ;
- les majuscules et les minuscules ne sont pas différenciées ;
- le nom de l'instance ne peut pas contenir les mots DEFAULT et MSSQLSERVER, ainsi que tout autre mot réservé ;
- le premier caractère du nom de l'instance doit être une lettre (A à Z) ou bien le trait de soulignement (_) ;
- les autres caractères peuvent être des lettres, des chiffres ou bien le trait de soulignement (_) ;
- les caractères spéciaux tels que l'espace, l'anti slash (\), la virgule, les deux points, le point-virgule, la simple cote, le "et" commercial (&) et l'arobase (@) ne sont pas autorisés dans le nom d'une instance.

### c. Le service SQL Server

Le moteur de base de données SQL Server s'exécute sous la forme d'un service Windows. Afin de pouvoir gérer le cas de plusieurs instances de SQL Server présentes sur le même serveur, chaque instance possède son propre service. Il est ainsi possible de gérer indépendamment l'arrêt et le démarrage de ces instances. Pour distinguer chaque instance au niveau des services, le service est nommé de la façon suivante MSSQL$nomInstance. Ainsi, pour l'instance SQLEXPRESS, le service se nomme MSSQL$SQLEXPRESS. Dans les différents outils graphiques de gestion des services, il sera présenté sous la forme suivante : SQL Server (SQLEXPRESS).

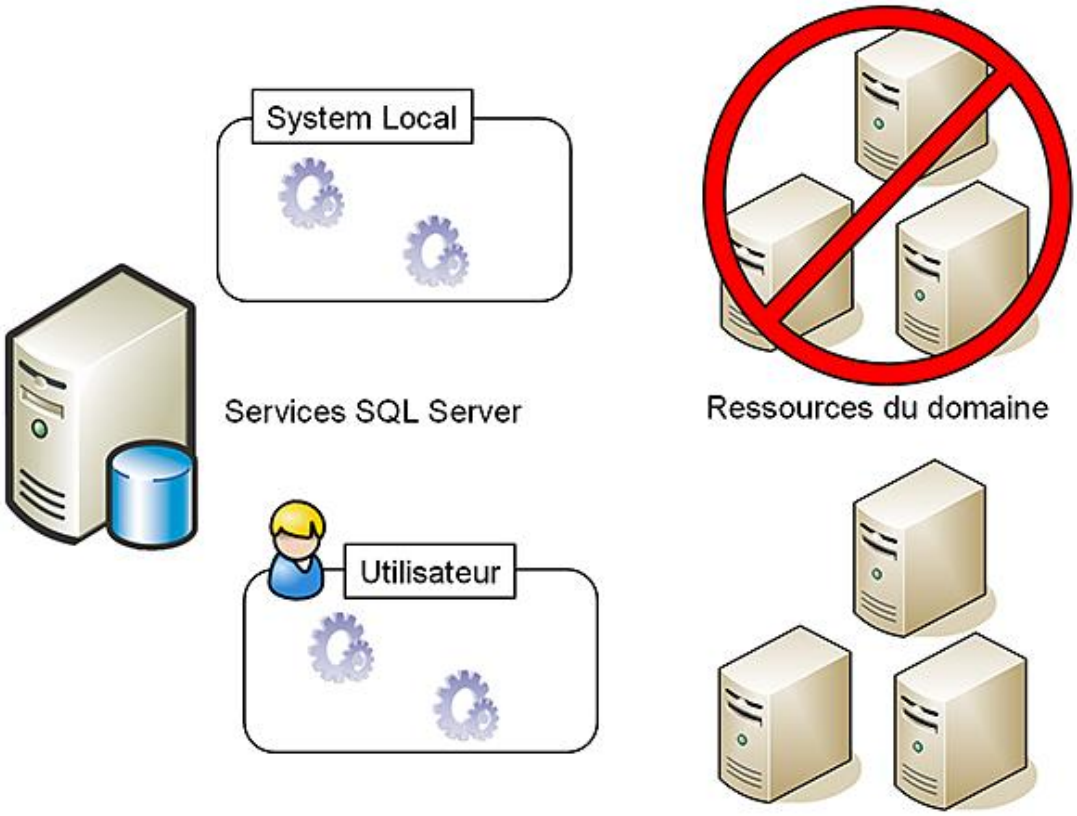
#### Compte Local Système et compte d'utilisateur

Les logiciels côté serveur s'exécutent sous forme de services. Comme tous les services, pour accéder aux ressources de la machine, ils utilisent le contexte d'un compte d'utilisateur du domaine. Par défaut les services s'exécutent dans le contexte du compte Local System. Ce compte permet d'obtenir toutes les ressources de la machine locale, mais il ne permet pas d'accéder à des ressources du domaine.

Le service d'exécution du moteur de base de données peut nécessiter d'accéder à des ressources présentes sur le réseau. Lors de l'installation, il est possible de préciser le compte d'utilisateur du domaine qui sera utilisé.

Pour simplifier les opérations de gestion, il est recommandé d'utiliser le même compte Windows pour les deux services.

Si l'entreprise comporte plusieurs serveurs SQL dans plusieurs domaines, il est préférable que tous les services SQL Server s'exécutent en utilisant un compte d'utilisateur de domaine portant le même nom et avec le même mot de passe.

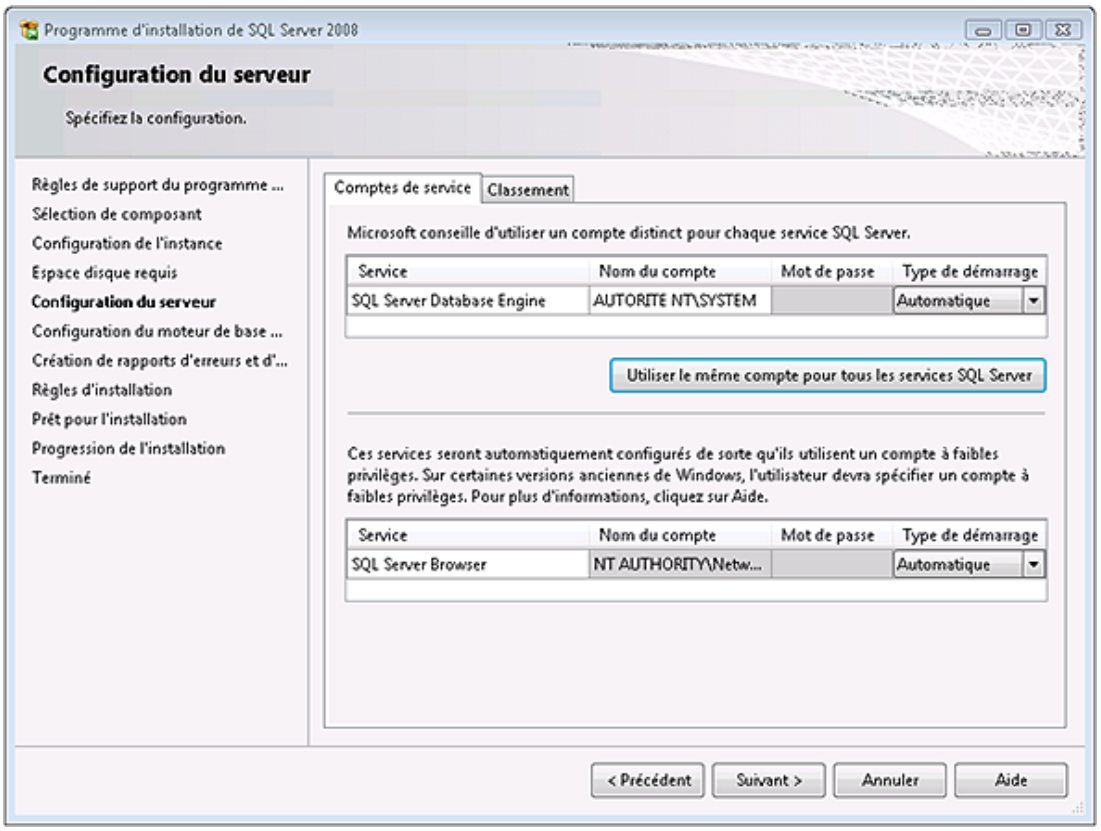



*Compte d'ouverture de session*

### Démarrage automatique du service

Il est possible de paramétrer le service de SQL Server de façon à ce qu'il démarre automatiquement lors du démarrage de Windows. L'avantage de cette option est qu'il n'est pas nécessaire d'ouvrir une session sur le poste en tant qu'administrateur, pour lancer les services du SGBDR. Lors de chaque démarrage de Windows, les bases gérées par le SGBDR sont immédiatement accessibles.

Ces deux options : **Comptes de services** et **Démarrage automatique de service** peuvent être fixées lors de l'installation de SQL Server ou bien une fois que SQL Server est installé au travers des outils d'administration.




### d. Paramètres de classement

La langue par défaut de l'instance de SQL Server va avoir une incidence directe sur le classement sélectionné.

Il est possible d'installer SQL Server quelle que soit la langue définie au niveau du système d'exploitation.

Il faut veiller à ne pas confondre la page de code et le classement.

La page de code est le système de codage des caractères retenus. La page de code permet d'identifier 256 caractères différents. Compte tenu de la diversité des caractères utilisés d'une langue à l'autre, il existe de nombreuses pages de code. Pour pouvoir prendre en compte des données exprimées dans différentes langues, il est possible d'utiliser le système UNICODE. Ce système de codage des caractères permet d'utiliser deux octets pour coder chaque caractère. Le système Unicode permet donc de coder 65536 caractères différents ce qui est suffisant pour coder la totalité des caractères utilisés par les différentes langues occidentales.

Cette solution avec les pages de codes n'est acceptable que si l'on stocke dans la base de données uniquement des données en provenance d'une seule langue. Or, avec la montée en puissance des applications de commerce électronique, les bases de données vont de plus en plus souvent contenir des informations (comme les nom, prénom et adresse des clients) en provenance de différentes langues. Pour pouvoir supporter les caractères spécifiques à chaque langue, il faut utiliser le type de données **Unicode**. Les données de type Unicode sont conservées dans les types **nchar** et **nvarchar**. Au niveau de l'application cliente, qui permet la saisie et l'affichage des informations, le type de données Unicode doit également être supporté.

---

L'espace réclamé par le type de données Unicode est deux fois plus important que pour les données non Unicode. Mais, ce léger désavantage est largement compensé par le temps gagné lors de l'affichage des données sur le poste client. En effet, avec le type de données Unicode, il n'est plus nécessaire de réaliser un mappage entre la page de code utilisée dans la base de données pour stocker l'information, et la page de code utilisée sur le poste client pour afficher l'information.

---

Le classement correspond à un modèle binaire de représentation des données et qui permet de définir des règles de comparaisons ainsi que des règles de tris. Par exemple, le classement permet de définir sur la langue française comment les caractères accentués doivent être pris en compte lors d'opération de comparaison et de tris.

Il existe par défaut trois types de classements :

- Les classements Windows qui s'appuient sur les paramètres de langues définies au niveau Windows. En s'appuyant sur ce type de classement, les ordres de tris, de comparaisons s'adaptent automatiquement à la langue du serveur.

- Les classements binaires sont réputés pour leur rapidité de traitement. Ils sont basés sur le code binaire utilisé pour enregistrer chaque caractère d'information au format unicode ou non.

- Les classements SQL Server sont présents pour assurer une compatibilité ascendante avec les versions précédentes de SQL Server. Il est donc préférable de ne pas faire ce choix dans le cadre d'une nouvelle installation.

Un classement doit nécessairement être précisé lors de la création de l'instance, il deviendra le classement par défaut de l'instance et sera utilisé par les bases de données **master**, **tempdb** et **model**.

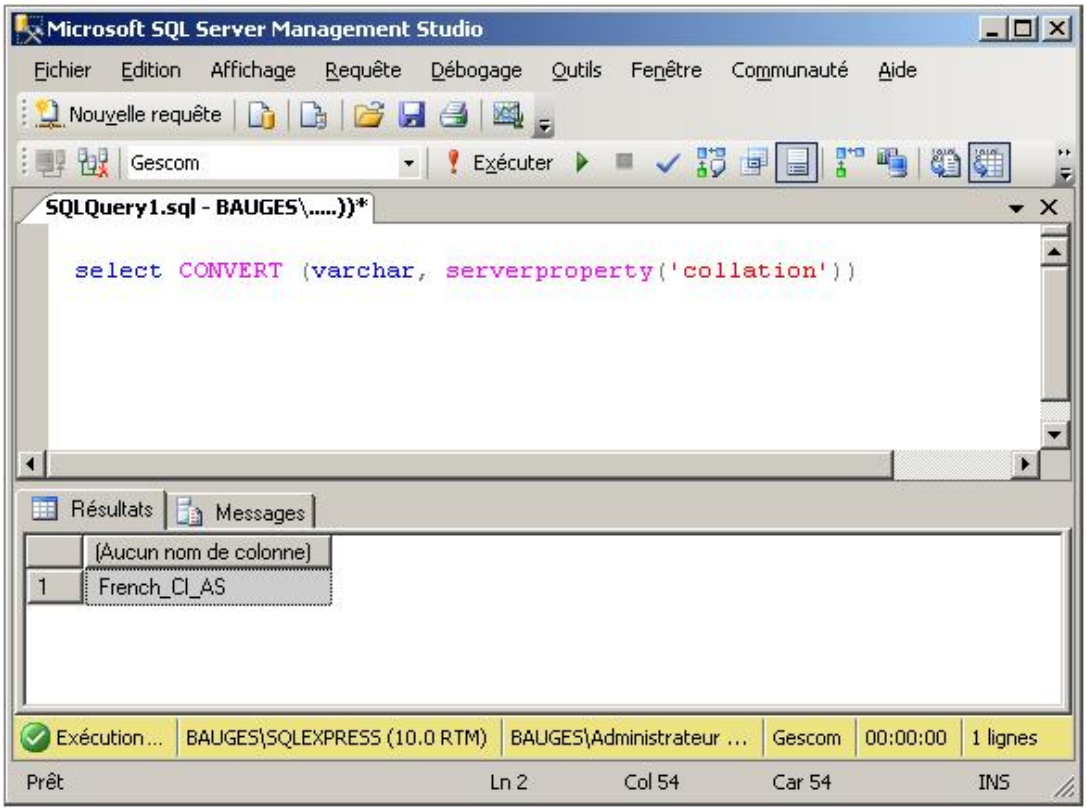
Lors de la création des bases de données utilisateurs, il sera possible de préciser un autre classement par l'intermédiaire de la clause **COLLATE**.

---

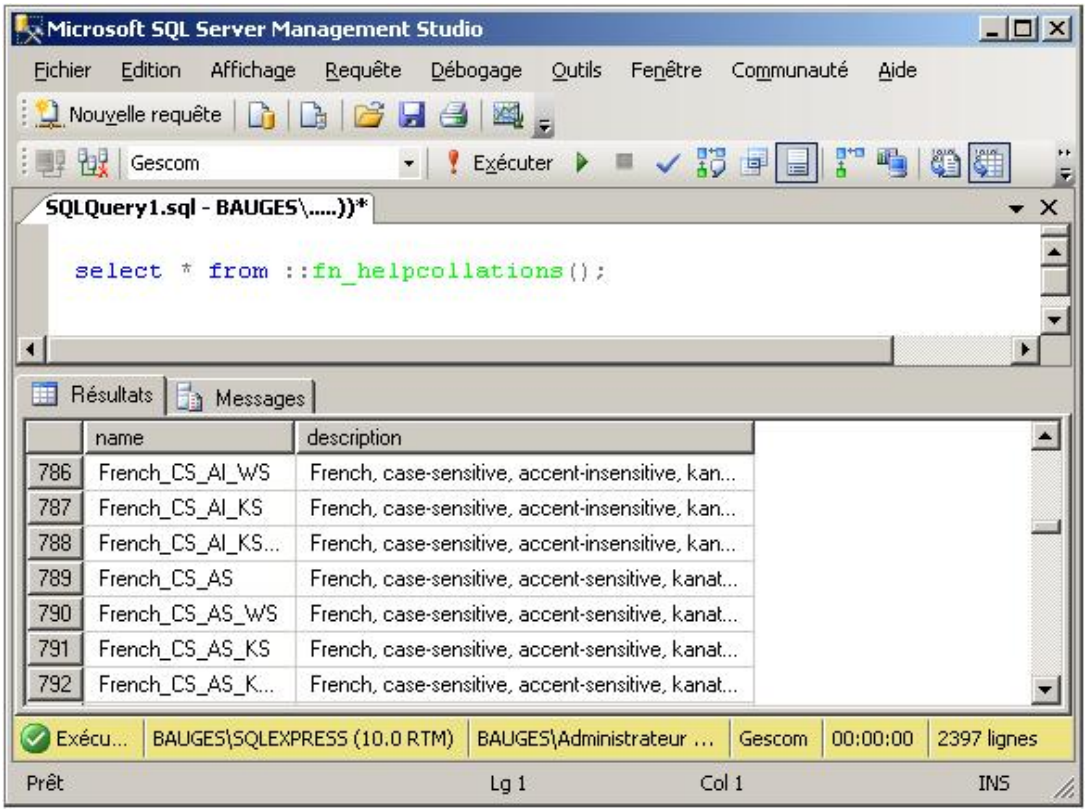
Comme les classements contrôlent l'ordre de tri des données Unicode et non Unicode, il n'est pas possible de définir des ordres de classement incompatibles.

---

Il est possible de connaître le classement adopté au niveau du serveur par l'intermédiaire de la fonction SERVERPROPERTY, comme illustré dans l'exemple ci-dessous :




Il est également possible de connaître l'ensemble des classements disponibles sur le serveur en utilisant la fonction : fn_helpcollations().




ATTENTION : le respect de la casse s'applique aussi bien aux identificateurs qu'aux données. Si l'on choisit l'ordre de tri binaire avec le respect de la casse, alors toutes les références aux objets doivent être réalisées en utilisant la même casse que celle spécifiée lors de la création de ces objets.

### e. Mode d'authentification

La gestion des comptes d'utilisateur peut s'appuyer intégralement sur les comptes des utilisateurs Windows mais il est aussi possible de définir des comptes d'utilisateur et de les gérer intégralement au sein de SQL Server. Dans ce cas, il est nécessaire de spécifier le mot de passe de l'utilisateur SQL Server qui possédera les privilèges d'administrateur SQL Server. Lorsque cela est possible, il est préférable de s'appuyer sur la sécurité Windows.

### f. Configuration du moteur de base de données

La configuration du moteur de base de données permet de spécifier le mode sécurité choisi, l'emplacement des fichiers de données, ainsi que l'activation ou non de l'option FILESTREAM.

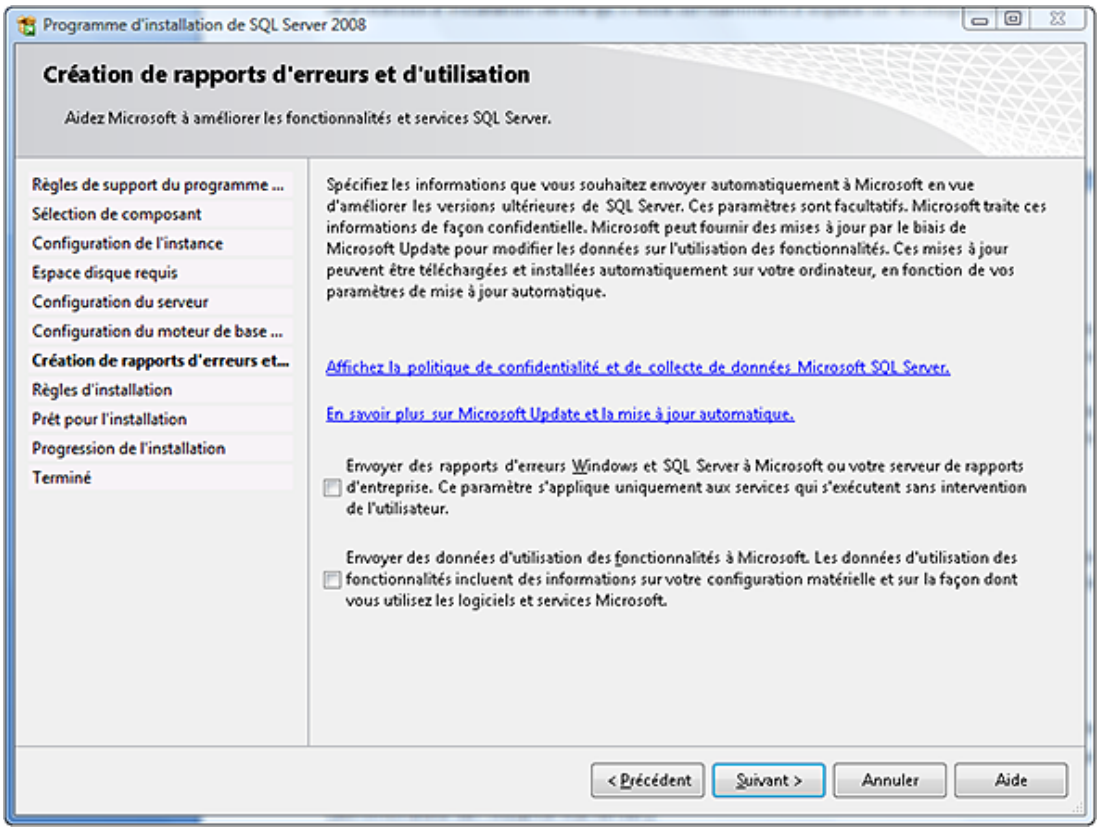
Depuis l'onglet **FILESTREAM**, il est possible d'activer cette fonctionnalité.

Ces différents choix peuvent être modifiés/outrepassés par la suite lors de la configuration du serveur ou bien lors de la gestion des fichiers relatifs à une base de données.

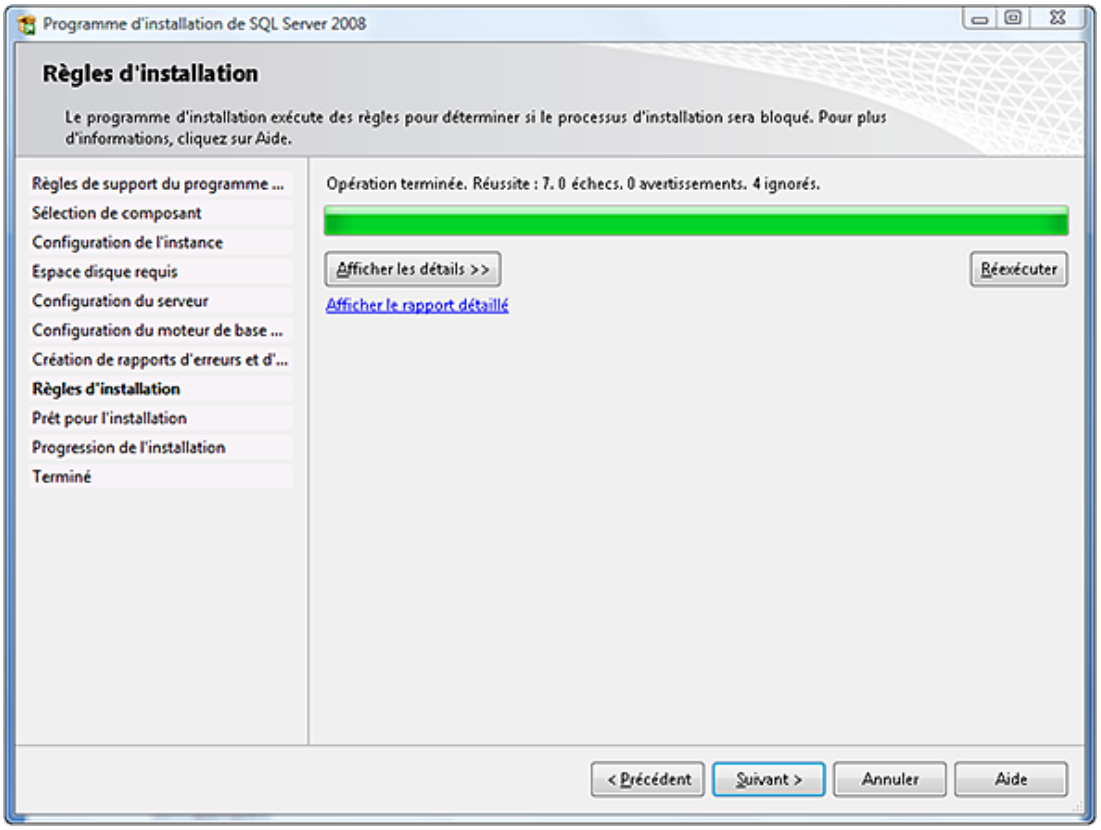
Pour une sécurité plus élevée au niveau des utilisateurs, il est recommandé de s'appuyer sur le contexte de sécurité Windows et d'interdire la sécurité SQL Server. Cela évitera, par exemple, que des développeurs codent en dur un nom et un mot de passe dans une application ou bien dans un fichier de configuration.

### g. Synthèse du processus d'installation

Le processus d'installation propose ensuite la création de rapports d'erreurs qui sont automatiquement envoyés à Microsoft.




Les règles d'installation sont vérifiées afin de s'assurer que rien ne viendra bloquer le processus d'installation.




## 3. Gestion du réseau

SQL Server utilise des bibliothèques réseau afin d'assurer la gestion de la transmission des paquets entre le serveur et le client. Ces bibliothèques réseau, existant sous forme de DLL (*Dynamic Link Library*), procurent toutes les

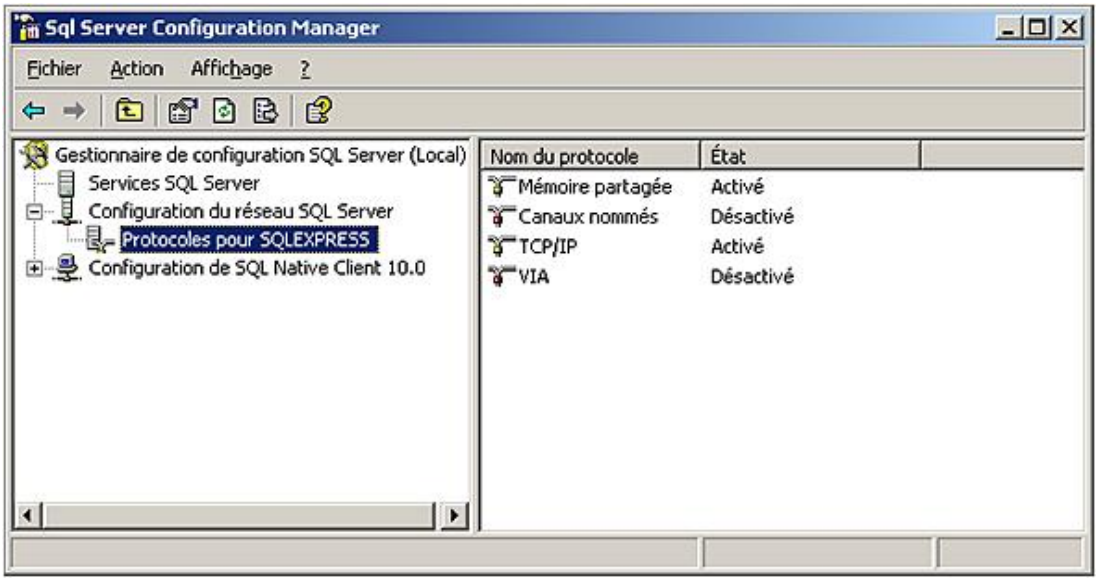
opérations nécessaires pour établir le dialogue entre le serveur et le client même si ces deux processus se trouvent sur le même poste.

Ces bibliothèques réseau sont utilisées par l'application via le mécanisme IPC ou Communication Inter Processus.

Un serveur peut être à l'écoute, simultanément, de plusieurs bibliothèques et accepte donc des demandes provenant de clients dialoguant avec des protocoles réseau différents. La seule obligation pour que le serveur puisse répondre au client est que la bibliothèque réseau correspondant à celle du client doit être installée sur le serveur.

Un fois que les bibliothèques réseau sont installées sur le serveur, il faut encore configurer les net library afin que le serveur puisse les prendre en compte.

La gestion du réseau entre le poste client et le serveur passe principalement par TCP/IP. C'est pourquoi la gestion de ce protocole est incluse par défaut lors de l'installation du serveur ou bien des utilitaires clients.




Bibliothèques disponibles :

**Canaux nommés**

Les canaux nommés sont désactivés pour toutes les éditions de SQL Server. Leur utilisation se limite au dialogue entre les outils graphiques et le service SQL Server sur le poste serveur.

**Sockets TCP/IP (par défaut)**

Cette net library permet d'utiliser les sockets Windows traditionnels.

Pour pouvoir utiliser correctement la net library TCP/IP, il est important de préciser le numéro de port auquel SQL Server répond. Par défaut, il s'agit du port 1433, numéro officiel attribué par l'IANA (*Internet Assigned Number Authority*) à Microsoft. Il est également possible d'utiliser un proxy. Dans ce cas c'est l'adresse du proxy qui sera précisée lors de la configuration de la net library TCP/IP.

Le port 1433 est utilisable par SQL Server si aucune autre application ou processus ne tente de l'utiliser simultanément.

Dans certains cas, comme l'accès au serveur au travers d'un pare feu, il est conseillé d'utiliser un port libre portant un numéro inférieur à 1024.

---

Dans le cas où SQL Server est configuré pour utiliser un port dynamique, le numéro du port est susceptible de changer à chaque démarrage de SQL Server. La gestion dynamique des ports TCP/IP est l'option de configuration à privilégier. Pour permettre au processus client d'adresser les demandes sur le bon port TCP/IP, le service SQL Server Browser se charge d'informer le processus client du numéro de port assigné à l'instance avec laquelle il souhaite travailler. Dans le cas où plusieurs instances SQL Server sont présentes sur le poste, ce service doit nécessairement être démarré.

---

Attention, bien que le protocole TCP/IP soit le moyen de communication couramment utilisé pour la communication entre différents postes via le réseau, ce protocole est désactivé par défaut. Il n'est donc pas possible à des processus externes de se connecter sur le serveur.

## 4. Exécution du programme d'installation

Le programme d'installation situé sur le CD-Rom de SQL Server s'exécute automatiquement dès que le CD-Rom est inséré dans la machine. Si la procédure d'installation ne démarre pas automatiquement, il convient de demander l'exécution du programme SETUP.EXE situé à la racine du CD-Rom.

## 5. Les bases d'exemples

Lors de l'installation du moteur de base de données, aucune base d'exemple n'est mise en place par défaut. En effet, sur un serveur de production, les bases d'exemples ne sont pas nécessaires et ne peuvent qu'introduire des failles de sécurité. Toutefois sur un serveur de test ou bien de formation, ces mêmes exemples prennent tout leur sens.

La documentation en ligne s'appuie sur la base AdventureWorks et ses diverses déclinaisons pour proposer des exemples illustrant les différentes instructions Transact SQL. Tous les exemples de code et de bases de donnés fournis par Microsoft sont disponibles sur codeplex (http://codeplex.com/SqlServerSamples). Ces exemples permettent d'illustrer de façon précise les différentes fonctionnalités offertes par SQL Server.

La base de données Northwind présente dans les versions précédentes de SQL Server est toujours disponible et peut ainsi être mise en place sur SQL Server 2008.

En fait, SQL Server 2008 ne propose pas une seule base d'exemple mais plusieurs en fonction de l'usage que l'on souhaite en faire. La base AdventureWorks2008 correspond à une base OLTP (*OnLine Transactional Processing*) classique et cette base illustre les utilisations classiques d'une base de données. La base AdventureWorksDW2008 est quant à elle une base qui illustre le datawarehouse (l'entrepôt de données) et sera utile lors de tests sur la réalisation, la gestion et la maintenance d'un entrepôt de données dans le cadre d'une analyse décisionnelle. La base AdventureWorksAS2008 est quant à elle axée sur Analysis Services. Enfin, une version allégée de la base OLTP est disponible sur la base AdventureWorkLT2008.

Lors du téléchargement de la base d'exemple souhaitée, ce n'est pas un script Transact SQL qui est enregistré sur le disque mais un fichier msi (SQL2008.AdventureWorks_OLTP_DB_v2008.x86.msi pour une plate-forme 32 bits). L'exécution de fichier permet une mise en place aisée de la base d'exemple.

Par défaut, et pour permettre un niveau de sécurité le plus élevé possible, le compte invité (guest) n'est pas défini, ceci afin d'empêcher les connexions anonymes. Dans le cadre d'un serveur de test, il peut parfois être intéressant d'autoriser ces connexions anonymes afin de permettre aux utilisateurs d'accéder librement à cette base d'exemple.

## 6. Le déploiement par xcopy

Il s'agit là de la solution la plus simple qui existe pour déployer des bases de données sous SQL Server Express. Dans ce cas, l'application .Net et le fichier de base de données (mdf) sont copiés directement sur l'ordinateur cible. Lors de l'exécution de l'application, SQL Server Express attache automatiquement le fichier mdf afin de rendre accessible la base. Bien entendu, l'opération inverse (détacher) est réalisée lorsque l'application cesse d'être exécutée. Pour que cette solution puisse être mise en place, il existe certaines restrictions.

Tout d'abord, il est nécessaire qu'une instance SQL Server Express soit déjà installée sur l'ordinateur cible. D'autre part, la base ainsi déployée ne doit pas être impliquée dans un processus de réplication. Enfin, les scripts Transact SQL ne doivent pas référencer le nom logique des fichiers de base de données. Cette restriction est due au fait que le nom logique des fichiers est construit automatiquement par SQL Server Express, lorsque la base est attachée, en s'appuyant sur l'emplacement physique actuel.

Pour que l'application puisse accéder à cette base qui est locale et qui est attachée/détachée de façon automatique par SQL Server Express, l'application doit veiller à utiliser une chaîne de connexion bien particulière :

- Dans le paramètre `DataSource`, il est nécessaire d'utiliser le `.` ou `local` à la place du nom de l'ordinateur soit `DataSource='.\SQLEXPRESS'` pour accéder à l'instance SQLEXPRESS locale.
- Le paramètre `Initial Catalog` ou `Database` doit être présent dans la chaîne de connexion mais ne doit pas être valorisé, soit `Initial Catalog=`.
- Le paramètre `AttachDBFile`, qui est optionnel, doit être spécifié et contenir le chemin d'accès au fichier mdf.

*Exemple de chaîne de connexion à utiliser par une application .Net :*

```
DataSource='.\SQLEXPRESS';
InitialCatalog=;
```

```
Integrated Security=true;
AttachDBFile='C:\ProgramFiles\MonProgramme\MaBase.mdf';
```

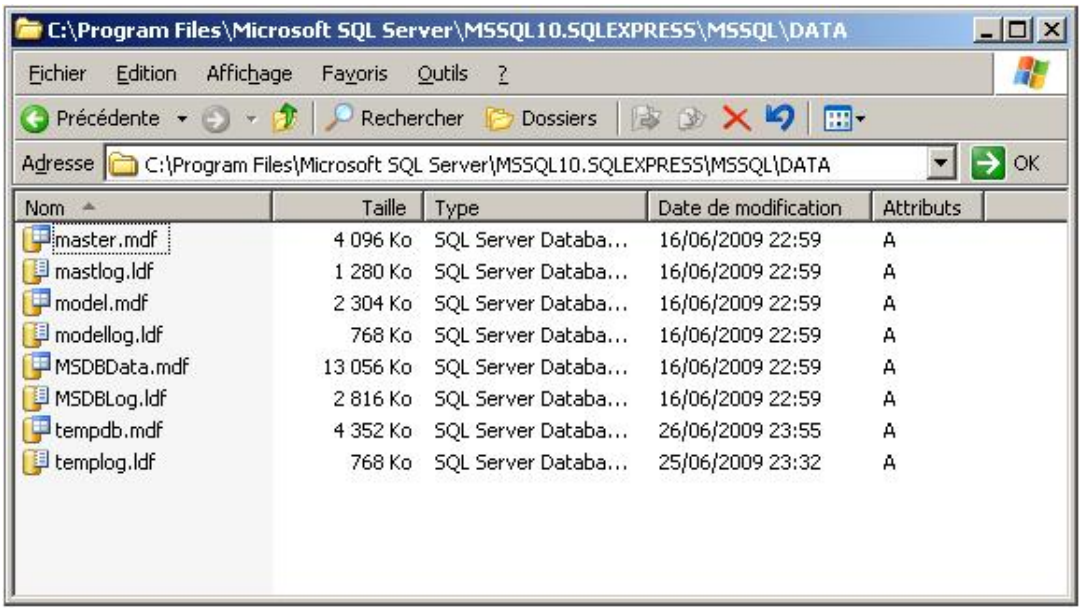
# Vérifier l'installation et configurer

## 1. Vérifier l'installation

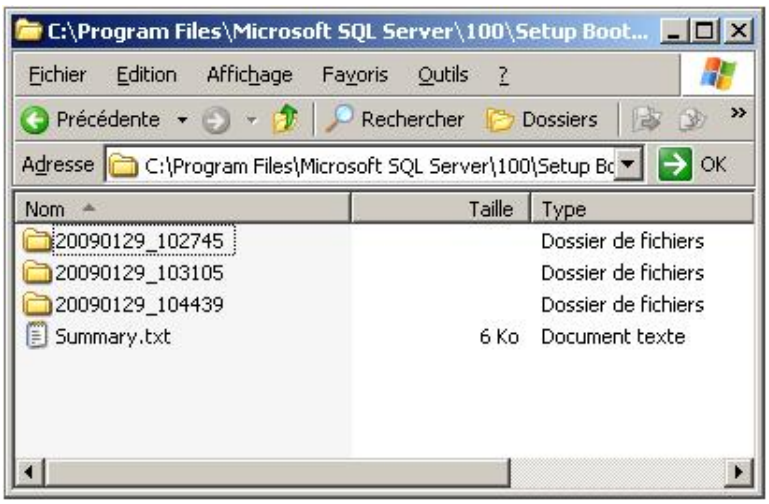
Comme pour tout produit, il est important de s'assurer que l'installation s'est correctement exécutée et que le serveur est maintenant opérationnel.

### a. Vérifier les éléments installés

Par l'intermédiaire de l'explorateur de fichiers, il est possible de vérifier que l'arborescence qui regroupe tous les fichiers nécessaires au bon fonctionnement du moteur de base de données est bien définie. Si l'installation est exécutée avec les paramètres par défaut, alors cette arborescence est c:\Program Files\Microsoft SQL Server\100. Les fichiers de données se trouvent quant à eux dans le dossier c:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\DATA sauf si un chemin autre a été défini lors de l'installation.




Au niveau des fichiers, il est possible de consulter le journal d'installation (Summary.txt) qui est présent dans le dossier c:\Program Files\Microsoft SQL Server\100\Setup Bootstrap\LOG. La lecture de document s'avère utile lorsque le processus d'exécution se termine en échec pour mieux cerner l'origine du problème. Ce fichier Summary est généré par chaque exécution du programme d'installation. Une nouvelle exécution du programme d'installation va renommer (avec horodatage) le fichier Summary existant afin de conserver la trace de toutes les installations.




Dans le cas où ce fichier Summary.txt ne vous fournit pas suffisamment d'information, il est possible de consulter le fichier SQLSetupxxx.cab situé dans le même dossier.

### b. Vérifier le démarrage des services

Pour un usage classique de la base, le service MS SQL Server doit être démarré et positionné en démarrage automatique. Le service MS SQL Server représente le moteur de la base de données et tant que ce service n'est pas démarré, il est impossible de se connecter au serveur et de travailler avec les données qu'il héberge. Pour gérer le service, il est bien sûr possible de le faire avec les outils proposés en standard par Windows, mais SQL Server propose également ses propres outils.

# Les outils

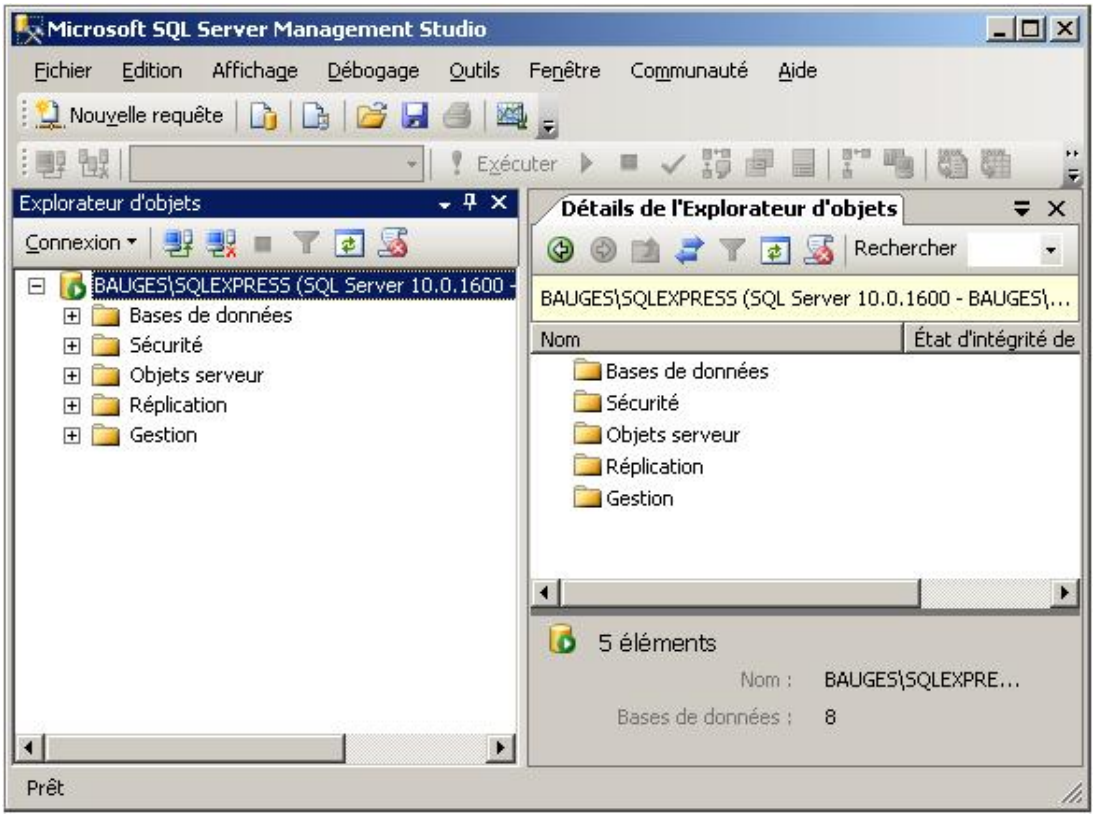
SQL Server dispose de nombreux outils. Ces outils sont complémentaires et chacun d'entre eux est adapté à un type de problème ou d'action. Il est important de posséder une vue d'ensemble sur l'objectif visé par chacun de ces outils afin de savoir lequel employer pour résoudre un problème bien précis. Il est possible de faire ici une analogie avec le bricolage, si vous ne possédez qu'un tournevis vous allez essayer de tout faire avec, alors qu'au contraire, si votre caisse à outils est bien garnie il y a de forte chance d'y trouver l'outil qui répond exactement au problème rencontré. Pour SQL Server l'approche est similaire. Il est bien entendu possible de faire quasiment tout en Transact SQL, mais SQL Server propose des outils graphiques autres pour permettre de solutionner des problèmes bien précis. L'utilisation de la plupart d'entre eux va être détaillée tout au long de cet ouvrage. Un regard global permet cependant d'avoir une meilleure vision de l'intérêt présenté par chacun.

Les outils ne sont présents que dans l'édition SQL Server Express Advanced.

## SQL Server Management Studio

Il s'agit de l'outil principal de SQL Server et il est destiné aussi bien aux développeurs qu'aux administrateurs.

SQL Server Management Studio (SSMS) est la console graphique d'administration des instances SQL Server. Il est possible d'administrer plusieurs instances locales et/ou distantes depuis cet outil. SQL Server Management Studio est également l'outil principal des développeurs de bases de données qui vont l'utiliser pour définir les scripts de création des tables, des vues, des procédures, des fonctions, des déclencheurs de base de données…



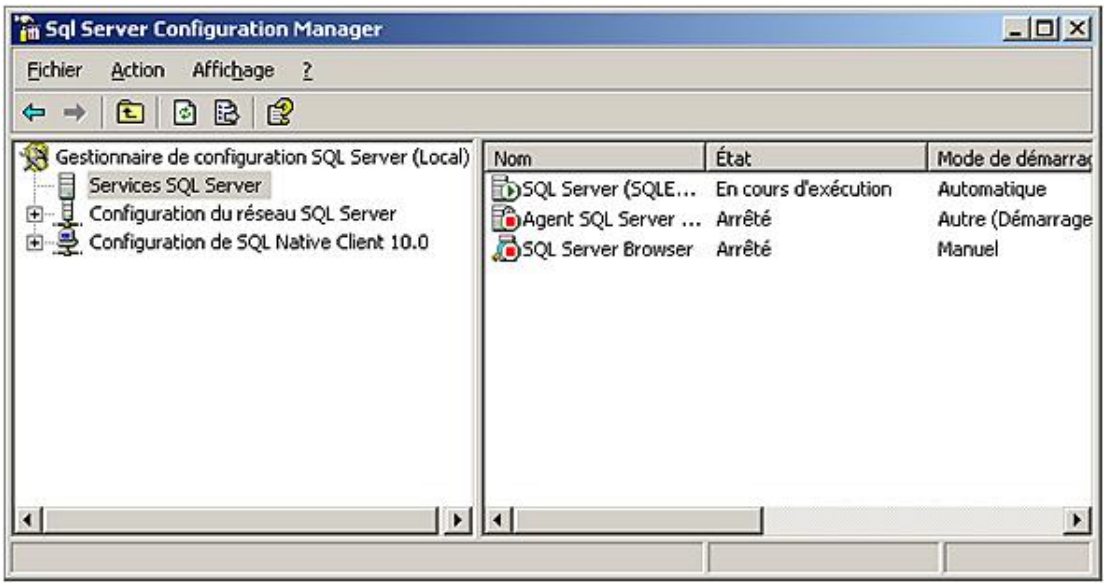

Cet utilitaire peut être démarré depuis la ligne de commande par l'intermédiaire de l'application **ssms**.

Avec SQL Server 2008, SSMS intègre son lot de nouveautés dont la très attendue autocomplétion qui permet le complément automatique des mots clés lors de l'écriture de requête. Cette autocomplétion est activée avec le traditionnel appui sur les touches [Ctrl][Espace]. En plus des mots clés, les références aux tables, colonnes, nom de procédures, fonctions sont disponibles au travers de l'autocomplétion. Cette fonctionnalité permet de gagner un temps précieux lors de l'écriture des scripts en limitant les erreurs de saisie.

Afin de faciliter la bonne gestion des bases de données, SQL Server Management Studio propose la génération de rapports. Ces rapports permettent d'avoir une vue globale et synthétique d'un ou de plusieurs éléments de la base ou du serveur. SQL Server 2008 propose un certain nombre de rapports prédéfinis mais il est possible de définir ses propres rapports en complément.

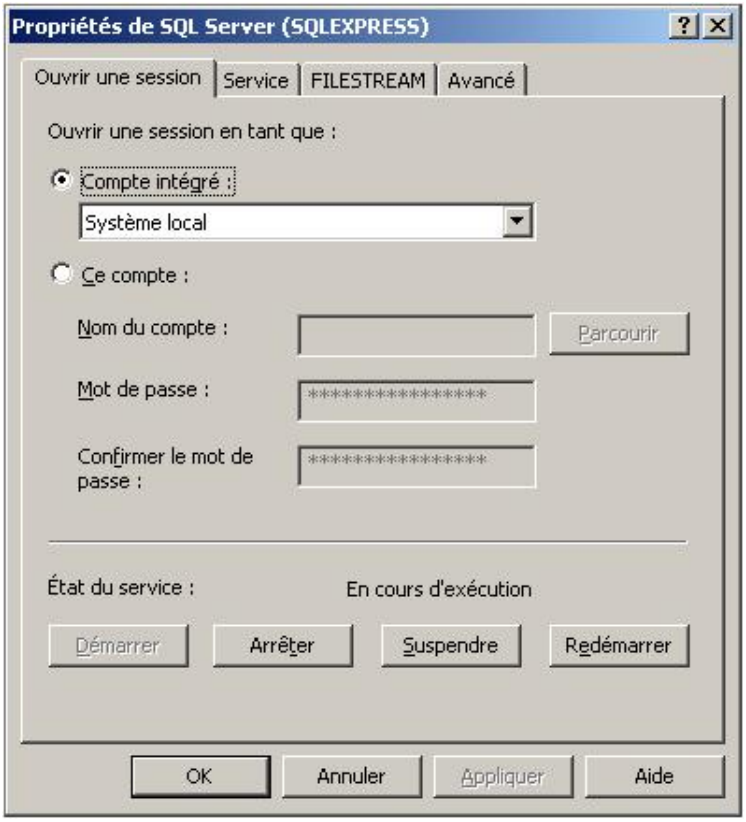
## Gestionnaire de Configuration SQL Server

Le gestionnaire de configuration de SQL Server permet de gérer l'ensemble des éléments relatifs à la configuration des services et du réseau côté client et côté serveur.



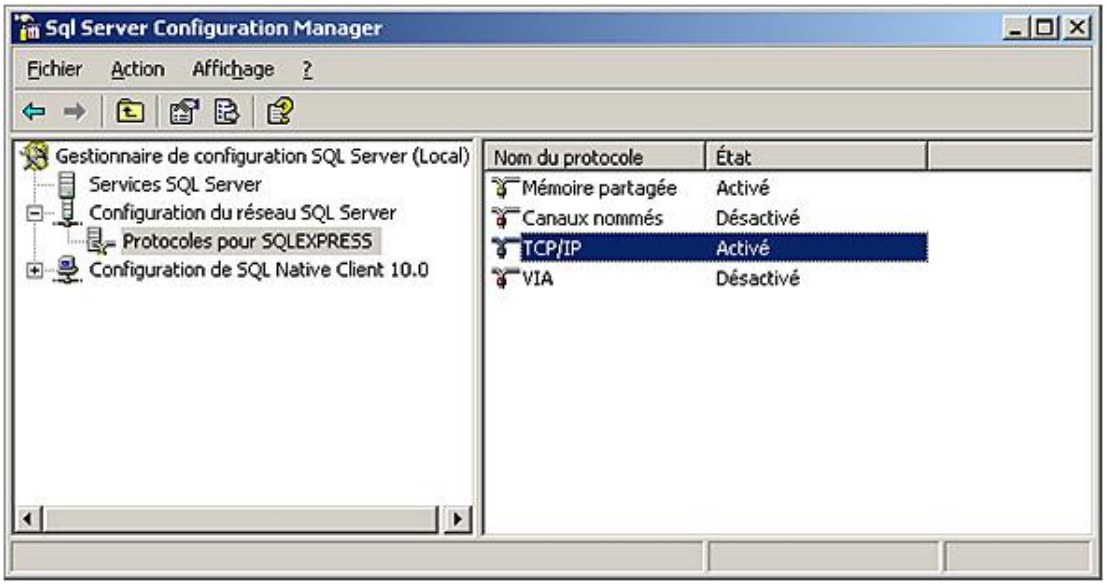

## La configuration des services

Les différents services relatifs à SQL Server peuvent être administrés directement depuis cet outil. En plus des opérations classiques d'arrêt et de démarrage, il est possible de configurer le type de démarrage (automatique, manuel, désactivé) ainsi que le compte de sécurité au sein duquel le service doit s'exécuter.



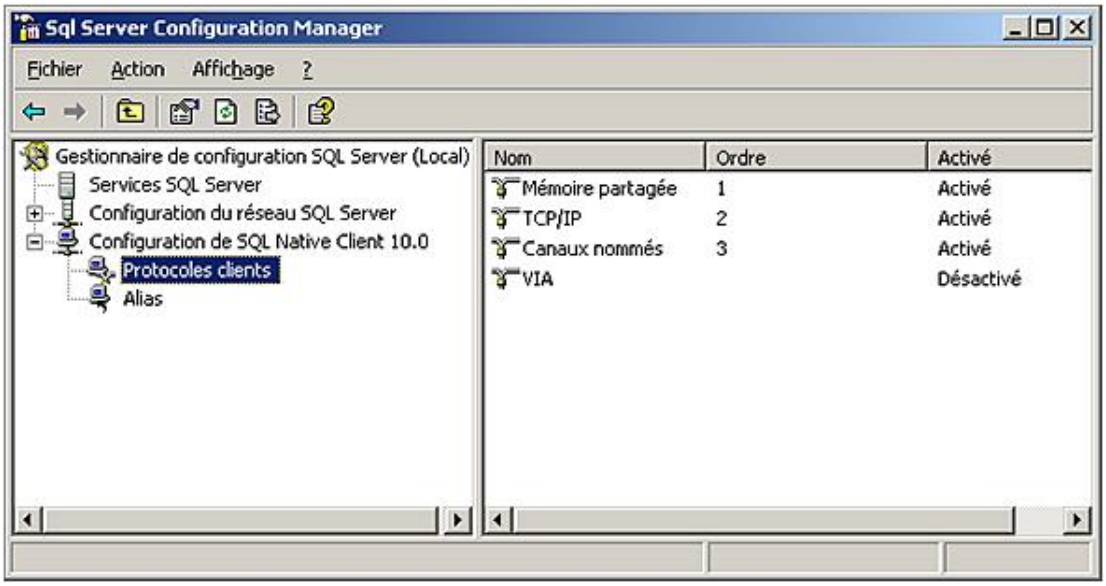

## Configuration de réseau SQL Server

Le gestionnaire de configuration permet également de gérer quels sont les protocoles pris en charge au niveau du serveur. Il est également possible à ce niveau de modifier les propriétés spécifiques à chaque protocole comme le numéro du port d'écoute du protocole TCP/IP.




## Configuration de SQL Native Client 10.0

Cette fois-ci la configuration porte sur les outils client installés localement et plus exactement de définir précisément les protocoles à leur disposition pour entrer en contact avec le serveur, mais aussi, lorsque cela s'avère nécessaire, la possibilité de définir des alias. Cette fonctionnalité est particulièrement intéressante lorsque le nom du serveur est enregistré dans une application et qu'il n'est pas possible ou bien facile de modifier cet enregistrement. La définition d'un alias permet de rediriger toutes les demandes vers un serveur distinct.




## SQLCmd

SQLCmd est un utilitaire en ligne de commande qui permet d'exécuter des scripts SQL. Cet outil va être mis à contribution lors de la demande d'exécution de tâches administratives concernant SQL Server à partir de Windows.

Cet outil permet de se connecter à une instance locale ou non de SQL Server. L'authentification à cette instance peut être effectuée en s'appuyant sur le mode de sécurité Windows ou bien SQL Server.

SQLCmd permet d'exécuter des scripts SQL que ce soit des instructions du DML (*Data Manipulation Language*) ou du DDL (*Data Definition Language*).

## osql

Autre outil en ligne de commande pour exécuter des scripts. Cet outil est maintenu pour des raisons de compatibilité mais étant donné qu'il s'appuie sur la technologie odbc, il est amené à disparaître.

## bcp

Il s'agit d'un utilitaire en ligne de commande qui permet d'extraire facilement et rapidement des données depuis la base vers un fichier ou bien de faire l'opération inverse.

### sqlps

Permet d'exécuter l'environnement PowerShell spécifique à SQL Server.

### sqldiag

Cet utilitaire de diagnostic peut être exécuté afin de fournir des informations au service support.

### sqllogship

Cette application permet de préparer un fichier journal par sauvegarde ou copie avant de l'envoyer vers une autre instance SQL Server.

## Utiliser un outil client pour se connecter à SQL Server

Pour établir la première connexion au serveur, plusieurs outils sont disponibles. En mode graphique, il convient de lancer SQL Server Management Studio. Au lancement de l'outil, la fenêtre suivante de connexion est présentée.


Il est possible d'établir la connexion depuis un outil en ligne de commande comme sqlcmd. L'écran suivant permet de se connecter au serveur en utilisant le mode de sécurité Windows.

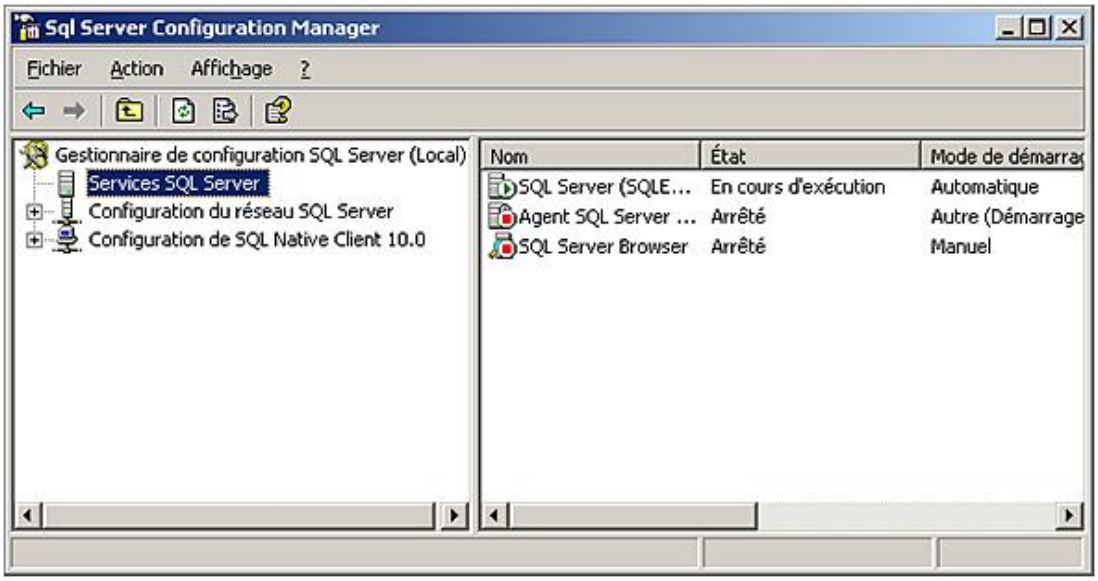
# La configuration

Avant de mettre en service le serveur SQL, en le rendant accessible par tous les utilisateurs, il est important de réaliser un certain nombre d'opérations de configuration du serveur et des outils d'administration client afin de se prémunir contre toute opération sensible.

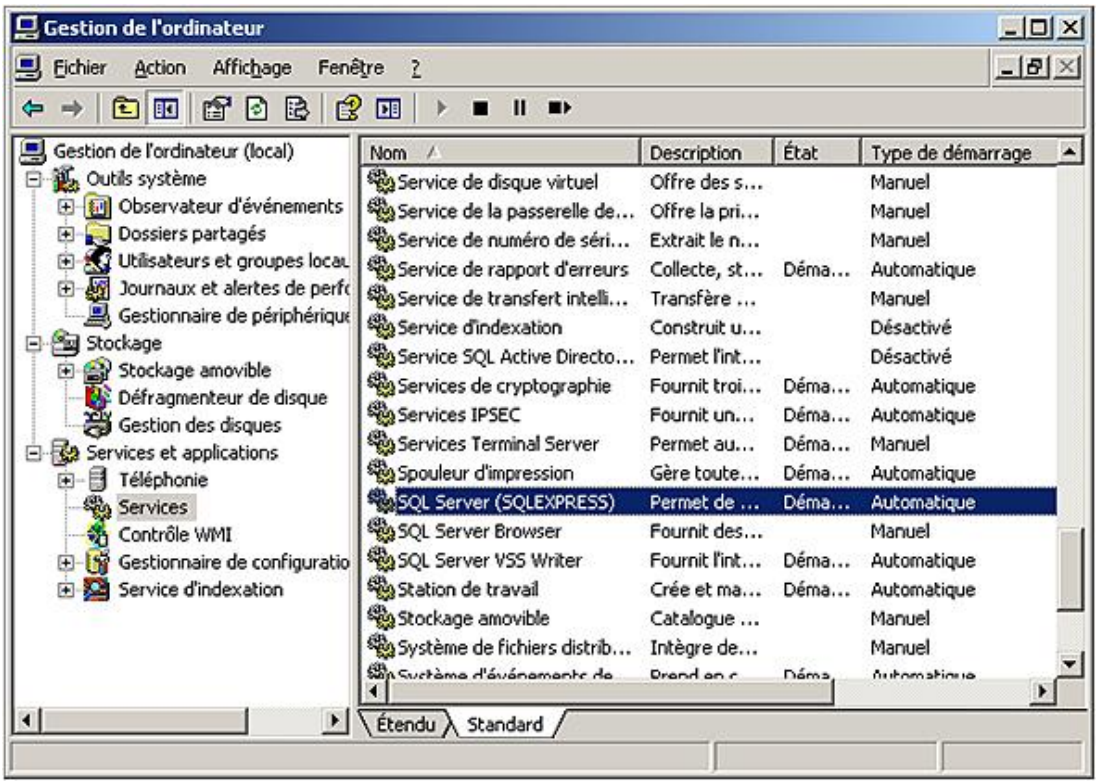
## 1. Les services

Les différents composants serveur s'exécutent sous forme de service. Il est donc nécessaire que ces services soient démarrés afin de pouvoir travailler avec le serveur. Ces services peuvent être gérés avec le gestionnaire de configuration de SQL Server mais ils peuvent également être gérés comme tous les services Windows.

Depuis le gestionnaire de configuration, il est simple de visualiser l'état du service ainsi que de modifier ses propriétés.




Comme tous les services Windows, ils peuvent être gérés de façon centrale au niveau du serveur Windows.



Enfin, il est possible d'agir sur ces services directement en ligne de commandes par l'intermédiaire des commandes **net start** et **net stop**. Lors d'un démarrage en ligne de commande, il est possible d'outrepasser la configuration par défaut du service en spécifiant la configuration à utiliser sous forme de paramètres. Par exemple, l'option m (net start mssqlserver m) permet de démarrer le serveur en mode mono utilisateur.

En cas de problème de démarrage, il est possible de démarrer le serveur SQL Server en tant qu'application à l'aide de l'exécutable sqlservr.exe. L'utilisation de cet exécutable permet de démarrer l'instance en ne tenant pas compte de toutes les options de configuration définies.

### Les différents états des services

**Démarré**

Lorsque le service MSSQL Server est démarré, les utilisateurs peuvent établir de nouvelles connexions et travailler avec les données hébergées en base. Lorsque le service SQL Server Agent est démarré, l'ensemble des tâches planifiées, des alertes et de la réplication est actif.

**Suspendu**

Si le service MSSQL Server est suspendu, alors plus aucun nouvel utilisateur ne peut établir une connexion avec le serveur. Les utilisateurs en cours ne sont pas concernés par une telle mesure. La suspension du service SQL Server Agent désactive la planification de toutes les tâches ainsi que les alertes.

**Arrêté**

L'arrêt du service MSSQL Server désactive toutes les connexions utilisateur et déclenche un processus de CHECKPOINT (l'ensemble des données validées présentes en mémoire sont redescendues sur le disque dur et le point de synchronisation est inscrit dans le journal). Un tel mécanisme permet d'assurer que le prochain démarrage du serveur sera optimal. Cependant le service attend que toutes les instructions en cours soient terminées avant d'arrêter le serveur. L'arrêt du service SQL Server Agent désactive l'exécution planifiée de toutes les tâches ainsi que la gestion des alertes.

## 2. SQL Server Management Studio

SQL Server Management Studio est l'outil de gestion graphique de SQL Server qui permet de réaliser les tâches administratives et toutes les opérations de développement. L'utilisation du même outil permet de réduire la distinction entre les deux groupes d'utilisateurs que sont les administrateurs et les développeurs. En partageant le même outil, il est plus facile de connaître ce qu'il est possible de faire d'une autre façon.

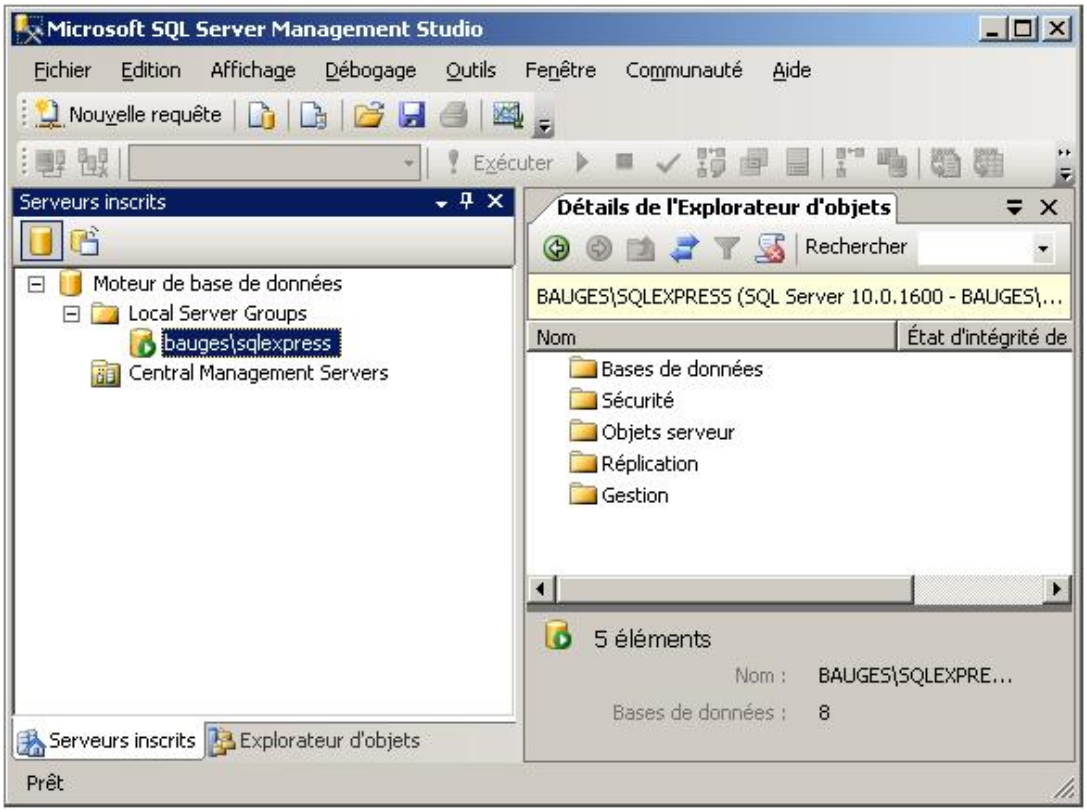
Pour pouvoir naviguer d'une instance à l'autre de SQL Server, éventuellement sur des serveurs différents, il est nécessaire d'enregistrer chaque serveur dans la console d'administration. Cette inscription n'est pas nécessaire pour l'instance locale de SQL Server, car lors de la création de l'instance, les informations relatives à cette instance ont été ajoutées dans SQL Server Management Studio.

C'est bien entendu une version simplifiée de SQL Server Management Studio qui est distribuée avec SQL Server Express. En effet, seules les actions directes sur le serveur sont possibles au travers de cette version de SQL Server Management Studio. Il n'est pas possible de gérer les services complémentaires tels que SQL Server Agent.

### Inscrire un serveur

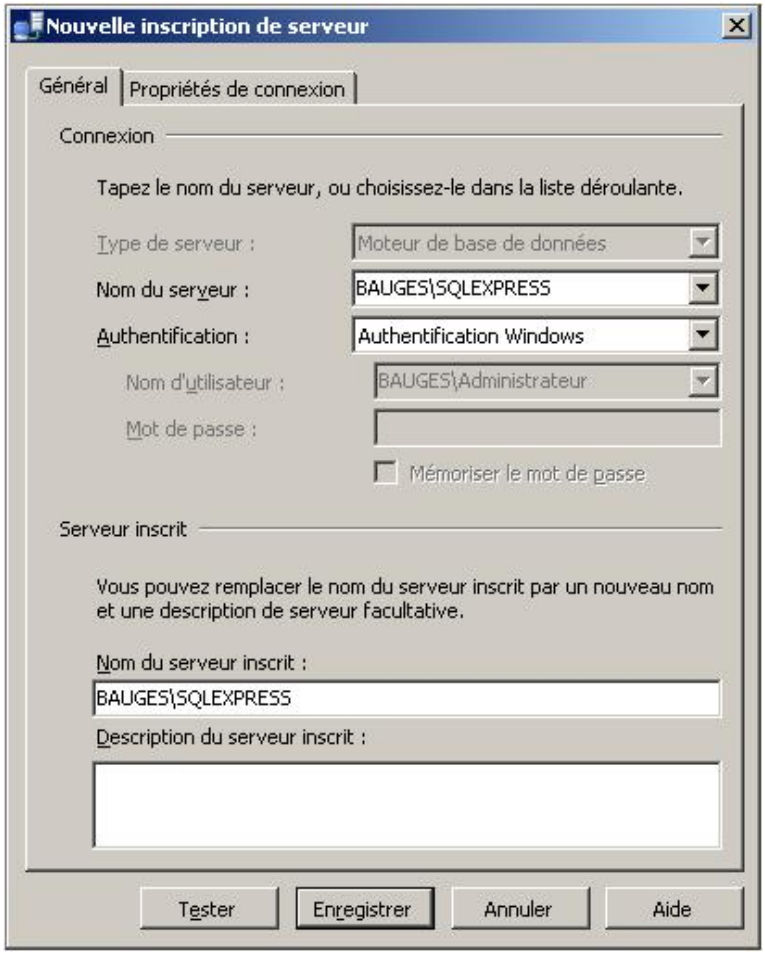
La fenêtre **Serveurs Inscrits** permet de connaître la liste des serveurs inscrits dans SQL Server Management Studio. Si cette fenêtre n'est pas visible, il est possible de demander son affichage par le menu **Affichage - Serveurs inscrits** ou par le raccourci clavier [Ctrl][Alt] **G**.

Les serveurs sont regroupés par type. Pour chaque type il est possible de définir des groupes de serveurs afin de les regrouper sur un autre critère, par exemple l'emplacement physique. Les groupes de serveurs n'ont aucune influence sur l'inscription du serveur. Il est possible de déplacer un serveur vers un groupe en sélectionnant l'option **Tâches - Déplacer vers** depuis le menu contextuel associé au serveur. Par analogie, il est possible de comparer les groupes de serveurs à des dossiers et les serveurs inscrits à des fichiers. Les fichiers ne sont pas affectés lorsqu'ils sont déplacés d'un dossier à un autre. Il en est de même pour les inscriptions de serveur. Les dossiers sont définis pour regrouper les fichiers suivant une certaine logique, c'est exactement le rôle que jouent les groupes de serveurs.




Pour inscrire un nouveau serveur depuis SQL Server Management Studio, il faut sélectionner l'option **Nouvelle inscription de serveur** depuis le menu contextuel associé au nœud **Local Server Groups** dans la fenêtre **Serveurs inscrits**.

La boîte de dialogue qui permet de réaliser l'inscription est composée de deux onglets. Le premier onglet permet de compléter toutes les informations générales de l'inscription comme le nom du serveur, mais également le type d'authentification utilisé pour établir la connexion sur le serveur.




Le bouton **Tester** permet de s'assurer que la connexion choisie permet bien de travailler sur le serveur sélectionné.

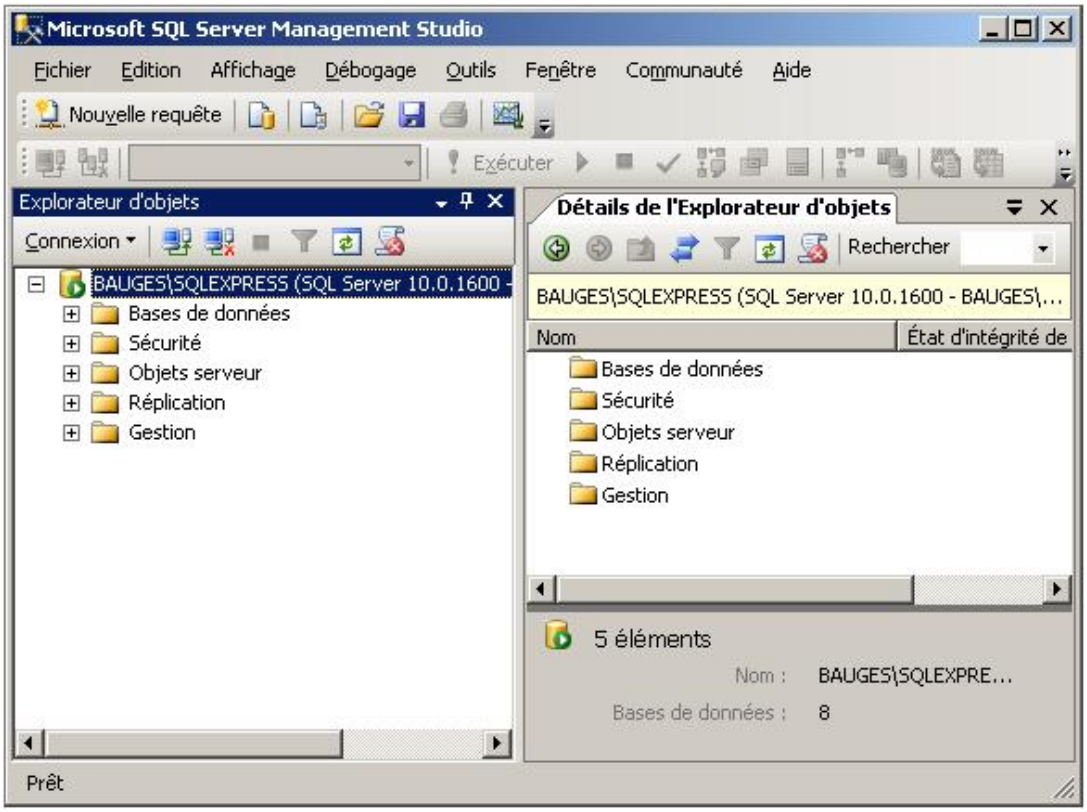
Il est possible d'enregistrer un serveur avec un nom différent de celui du serveur.

Le second onglet permet, quant à lui, de fixer des options plus avancées comme la base de données par défaut ou bien le type de protocole réseau utilisé pour établir la connexion avec le serveur.

Pour des raisons de sécurité, il est préférable de choisir lorsque cela est possible le mode d'authentification Windows.

Lors de l'inscription du serveur, il possible de sélectionner certaines options comme le délai d'expiration de la connexion.


Pour faciliter le travail de l'administrateur, les bases de données système sont regroupées dans un dossier. Ainsi, ce sont les bases de données des utilisateurs qui sont visibles au premier plan. Cette séparation permet de ne pas encombrer la console par les bases système qui n'ont d'importance que pour SQL Server.




Le même type de séparation est effectué sur les bases entre les tables système et les tables créées par les

utilisateurs. Ce sont ces dernières qui contiennent les informations et sur lesquelles tous les efforts d'administration vont porter.

## 3. Configuration du serveur

Avant d'ouvrir plus librement l'accès au serveur et de permettre aux utilisateurs de venir travailler sur le serveur, il convient de surveiller attentivement les deux points suivants :

### Mot de passe de l'administrateur

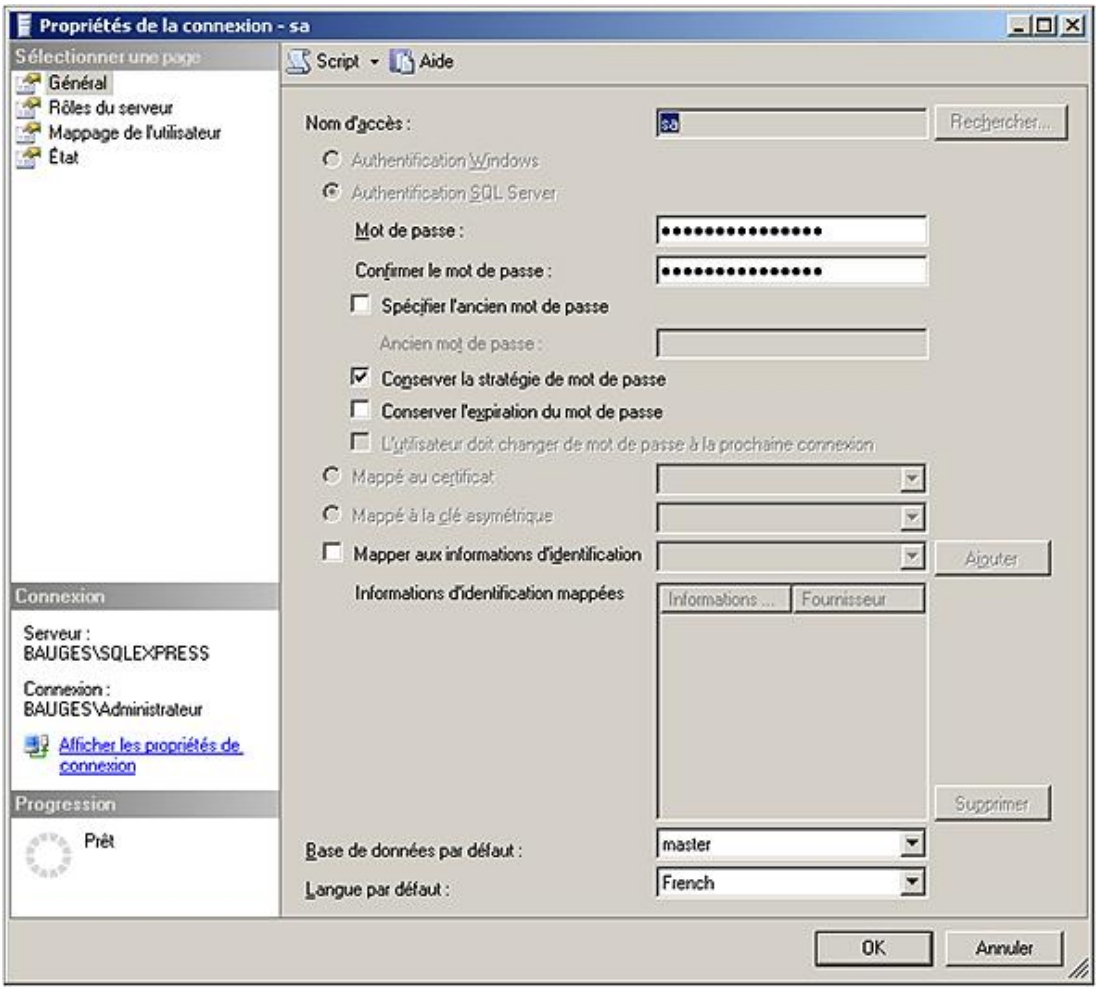
Cette préoccupation concerne uniquement les serveurs qui sont configurés en mode de sécurité mixte. Si ce choix a été fait au cours de l'installation, il faut s'assurer que le mot de passe de l'administrateur SQL Server (sa) est suffisamment fort. Si ce n'est pas le cas, il est alors nécessaire de le modifier.

Lors de l'installation de SQL Server deux utilisateurs sont prédéfinis. Le premier est le groupe local des administrateurs (utilisé avec la sécurité Windows), le second est l'utilisateur **sa**. Ces deux utilisateurs présentent des droits d'administrateur du serveur SQL. L'utilisateur **sa** s'appuie sur le sécurité SQL Server et son mot de passe a été demandé durant la procédure d'installation.

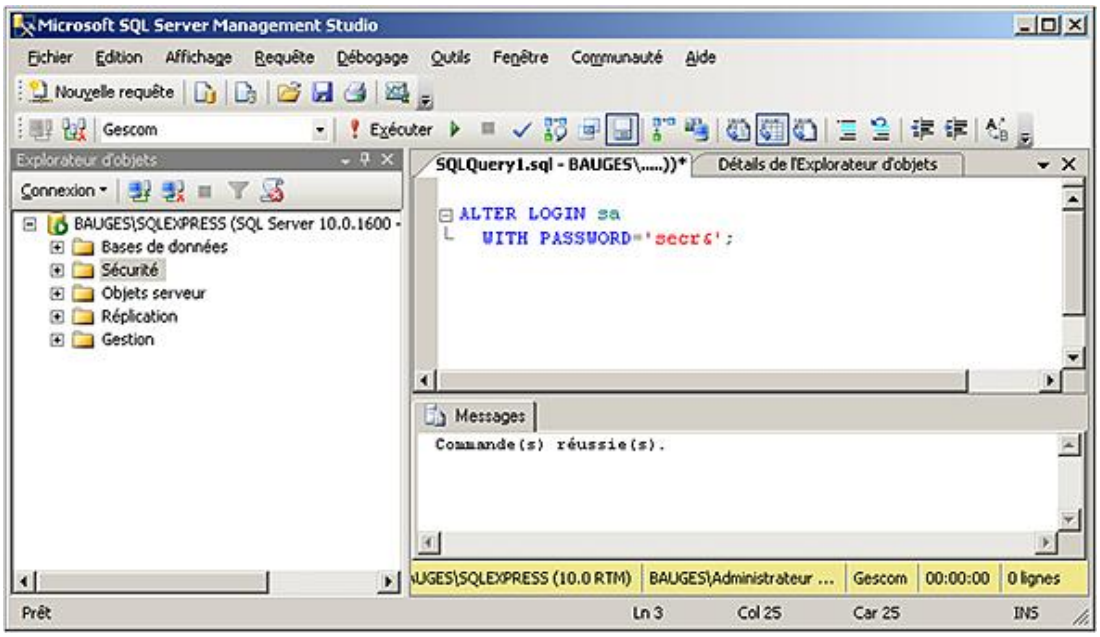
Si lors de l'installation, seul le mode de sécurité Windows a été activé, alors la connexion **sa** est non active. Il est nécessaire de définir un mot de passe fort avant d'activer la connexion. Cependant la connexion ne pourra être utilisée que si le serveur est configuré en mode de sécurité mixte.

Deux possibilités se présentent.

#### Par SQL Server Management Studio

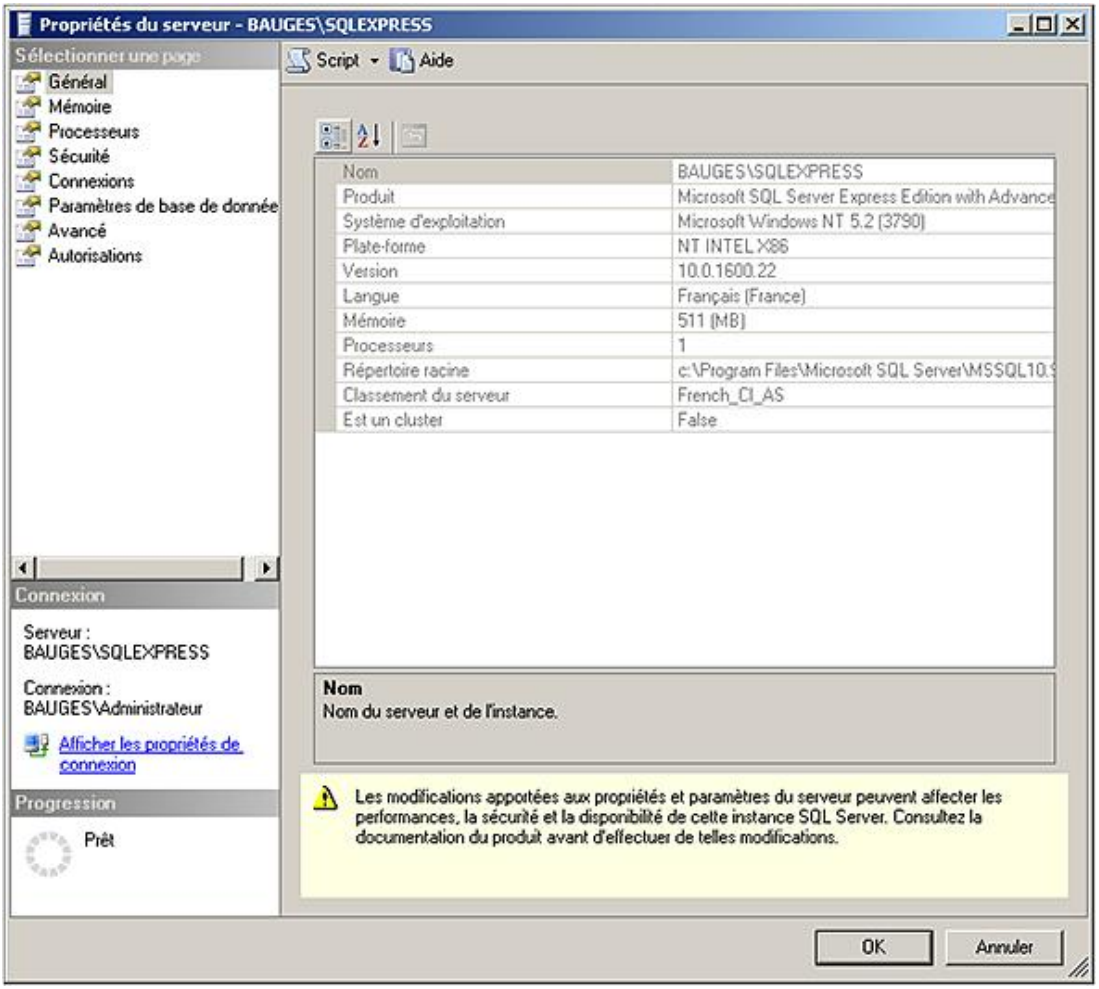



#### Par le Transact SQL

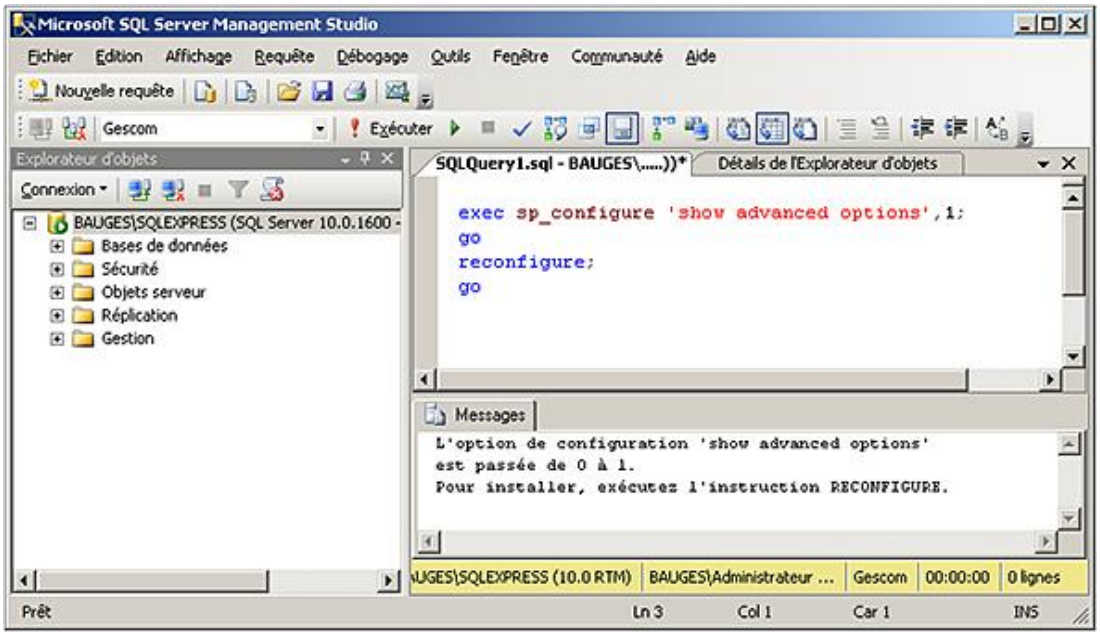



## Gestion des ressources

Les ressources de la machine sont gérées dynamiquement. Cette gestion automatique des ressources permet d'offrir les meilleures fonctionnalités possibles du serveur tout en réalisant un minimum de tâches administratives. Toutefois, il peut parfois être intéressant de gérer manuellement certaines ressources afin de réaliser une optimisation de l'utilisation des ressources du serveur. Quelques-uns de ces réglages peuvent être réalisés au moyen de SQL Server Management Studio.




Pour accéder à la totalité des paramètres du serveur, il faut utiliser la procédure stockée **sp_configure**.




Si une option est modifiée à l'aide de la procédure **sp_configure**, elle ne prendra effet que lors du prochain redémarrage du serveur SQL. Il est possible d'appliquer la modification de façon immédiate en exécutant la commande **RECONFIGURE WITH OVERRIDE**.

## 4. La gestion du processus SQL Server

Pour le système d'exploitation, chaque application s'exécute sous forme de processus. Chaque processus dispose de ces propres threads qui correspondent aux unités de travail que le système d'exploitation doit soumettre au processeur. A un processus correspond toujours au moins un thread.

Chaque instance SQL Server gère elle-même ses propres threads et gère leur synchronisation sans passer par le noyau Windows. L'objectif de SQL Server est de répondre efficacement et rapidement à des demandes de montées en charge brusque et soudaine. Afin d'être toujours disponible, SQL Server gère son propre pool de threads dont le nombre maximal est contrôlé par le paramètre **max worker threads**. Avec la valeur par défaut (0) SQL Server se charge de gérer lui-même ce nombre de threads mais il est également possible de fixer le nombre maximum de threads. Ces threads vont avoir pour objectif de traiter les requêtes des utilisateurs. Étant donné qu'un utilisateur ne travaille pas 100 % de son temps sur le serveur, il doit lire ou bien modifier les données avant de soumettre une nouvelle requête, il est préférable pour SQL Server de partager un même thread entre plusieurs utilisateurs.

---

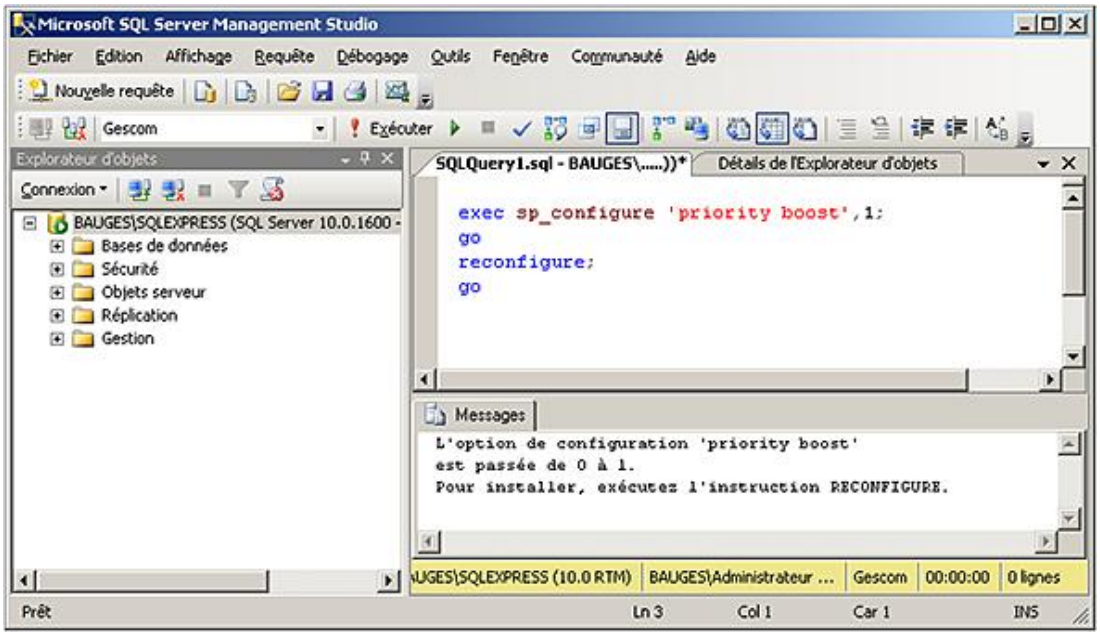
La valeur maximale de ce paramètre était de 255 avec SQL Server 2000. Aussi, dans le cadre d'une migration de serveur, est-il recommandé de positionner cette valeur à 0.

---

Windows propose également de gérer des fibres qui représentent une unité de travail plus légère qu'un thread. Il est possible de demander à SQL Server de travailler avec ces fibres en lieu et place des threads. Ce paramétrage s'effectue avec l'option de configuration **lightweight pooling**. Cependant, l'activation de cette option rend impossible l'utilisation de code CLR dans SQL Server. D'autre part, l'activation de cette option se traduira par un gain significatif de performance uniquement sur des serveurs massivement multiprocesseurs avec un taux d'utilisation des processeurs important.

Dans le cadre d'une architecture multiprocesseurs et souvent multi-instance, il est possible à l'aide de l'option de configuration **affinity** de spécifier les processeurs à utiliser pour chaque instance. Cette option contient une valeur binaire ou chaque bit représente l'autorisation (1) ou non (0) d'utiliser un processeur.

Enfin pour ordonnancer l'exécution des différents threads, Windows affecte à chaque processus une priorité variant de 1 (le moins prioritaire) à 31 (le plus prioritaire). Cette gestion de priorité ne concerne pas les processus système. Par défaut, le processus SQL Server reçoit un niveau de priorité égal à 7, c'est-à-dire un processus normal. Il est possible de donner une priorité supérieure par l'intermédiaire de l'option de configuration **priority boost**. Cette option peut s'avérer particulièrement intéressante lorsque plusieurs instances SQL Server s'exécutent sur le même poste et que l'on souhaite en favoriser une.




## 5. La gestion de la mémoire

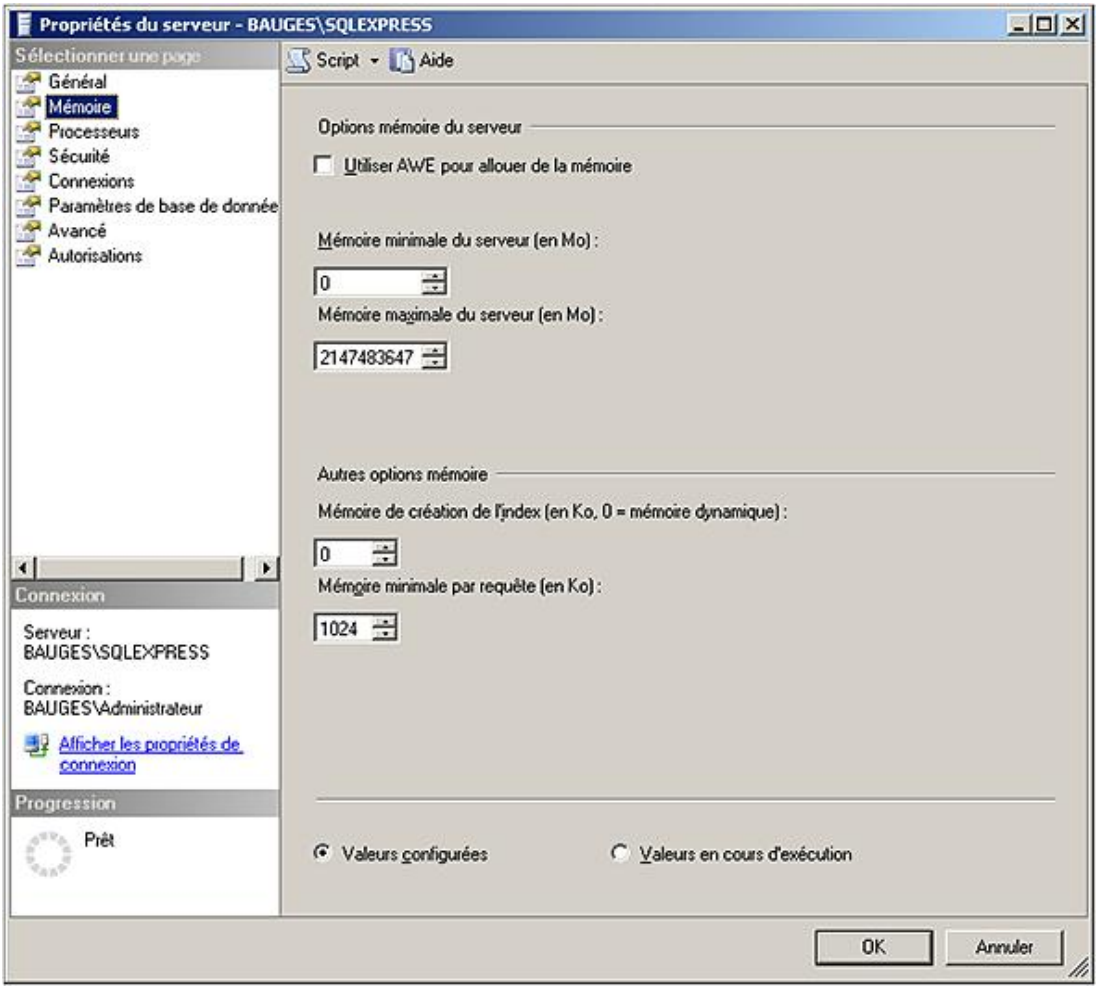
Par défaut, SQL Server gère dynamiquement la quantité de mémoire dont il a besoin. Il est d'ailleurs recommandé de conserver une gestion dynamique de la mémoire qui permet une répartition optimale de la mémoire entre les différents processus s'exécutant sur le serveur.

Il est important que le serveur dispose d'une quantité de mémoire suffisante car cela permet de minimiser le nombre de lectures physiques et de favoriser les lectures logiques. Plus ces dernières sont nombreuses, meilleurs sont les temps de réponse du serveur. Le ratio entre ces deux types de lectures peut être obtenu au travers de l'analyseur de performances.

Pour permettre la gestion dynamique de la quantité de mémoire utilisée, SQL Server s'appuie sur l'API (*Application Programming Interface*) de gestion de la mémoire de Windows afin d'acquérir le maximum de mémoire sans pour autant privé le système de la quantité de mémoire qui lui est nécessaire.

Cette gestion dynamique peut être limitée en utilisant les paramètres **min Server memory** et **max Server memory**. L'instance SQL Server, même si elle est peu utilisée, conservera toujours la quantité de mémoire spécifiée par **min Server memory**. En cas de charge de travail, il est possible d'acquérir de la mémoire, sans jamais dépasser la valeur spécifiée par **max Server memory**.

Une instance basée sur l'édition Express n'est pas en mesure d'exploiter plus de 1 Go de mémoire physique.



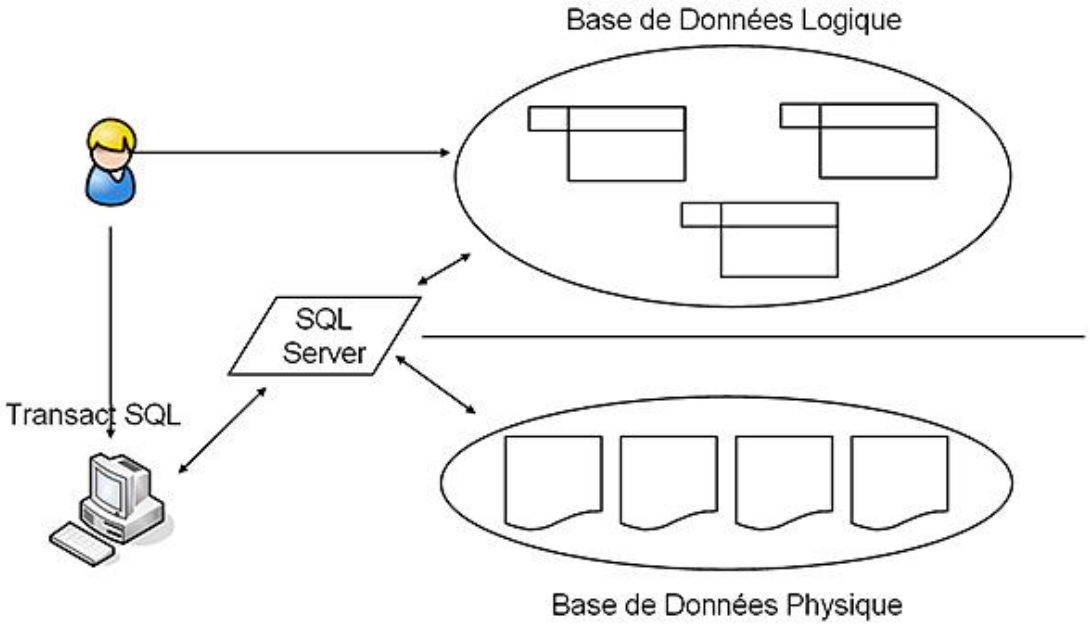
# Notions générales

L'installation du serveur SQL réalisée, il convient de définir des espaces logiques de stockage afin de regrouper sous un même nom l'ensemble des données correspondant à un même projet. Cet ensemble est la **base de données**, elle va nous permettre de travailler logiquement avec des objets tels que les tables sans jamais avoir à se soucier du stockage physique. SQL Server permet de réaliser des associations entre les fichiers physiques et les bases de données. Dans ce chapitre, la création et la gestion des fichiers physiques seront abordées en même temps que les bases de données.

## 1. Liens entre base de données et organisation physique

Lors de la création d'une base de données, il est nécessaire de préciser au moins deux fichiers. Le premier servira à stocker les données, le deuxième sera utilisé par le journal afin de stocker les images avant et après modification des données.

Ces deux fichiers sont obligatoires et sont propres à chaque base. Dans SQL Server, il n'est pas possible de partager un fichier de données ou le journal entre plusieurs bases.




*Séparation entre les schémas logique et physique*

## 2. La notion de transaction

### a. Qu'est-ce qu'une transaction ?

Une transaction est un ensemble indivisible d'ordres Transact SQL. Soit la totalité peut s'exécuter, soit aucun ordre ne peut s'exécuter. Le moteur SQL doit être capable, tant que la transaction n'est pas terminée, de remettre les données dans l'état initial. Si la transaction n'est pas terminée, aucun autre utilisateur ne peut intervenir sur les données, tant en lecture qu'en écriture. Il est dangereux de s'appuyer sur des données, dont on ignore si les modifications en cours vont persister dans le temps ou non. Afin de garantir la cohérence des données, toutes les lignes qui sont modifiées à l'intérieur d'une transaction sont verrouillées pour qu'aucun autre utilisateur ne puisse intervenir sur ces lignes. Le verrouillage est effectué automatiquement, et les verrous sont relâchés lorsque l'utilisateur indique la fin de la transaction, soit en succès, soit en échec. Le verrouillage des données est géré de façon optimale par SQL Server de façon à minimiser le nombre de lignes de données verrouillées (verrouillage au niveau de la ligne possible) ainsi que le nombre de verrous posés (verrous de ligne, de blocs ou de table). Lorsqu'une donnée est verrouillée et qu'une transaction autre que celle qui a posé le verrou la réclame, elle doit attendre la libération du verrou pour accéder à l'information. Les files d'attente pour l'accès aux données sont gérées par des listes FIFO (Premier Entré Premier Sorti).

**Exemple de transaction** : un exemple connu mais représentatif de la notion de transaction est celui du retrait d'argent auprès d'un distributeur automatique. La transaction est alors constituée de deux opérations : le débit du compte et la distribution d'argent. S'il n'est pas possible de réaliser l'une des deux opérations, c'est l'ensemble des opérations qui devra être annulé. Il paraît déraisonnable de débiter le compte si la somme correspondante n'a pas

été distribuée à l'utilisateur.

## b. Les ordres Transact SQL

Ils sont au nombre de quatre, BEGIN TRAN, COMMIT TRAN, ROLLBACK TRAN, SAVE TRAN qui indiquent respectivement le début de la transaction, la fin avec succès, la fin avec échec et la définition des points d'arrêt.

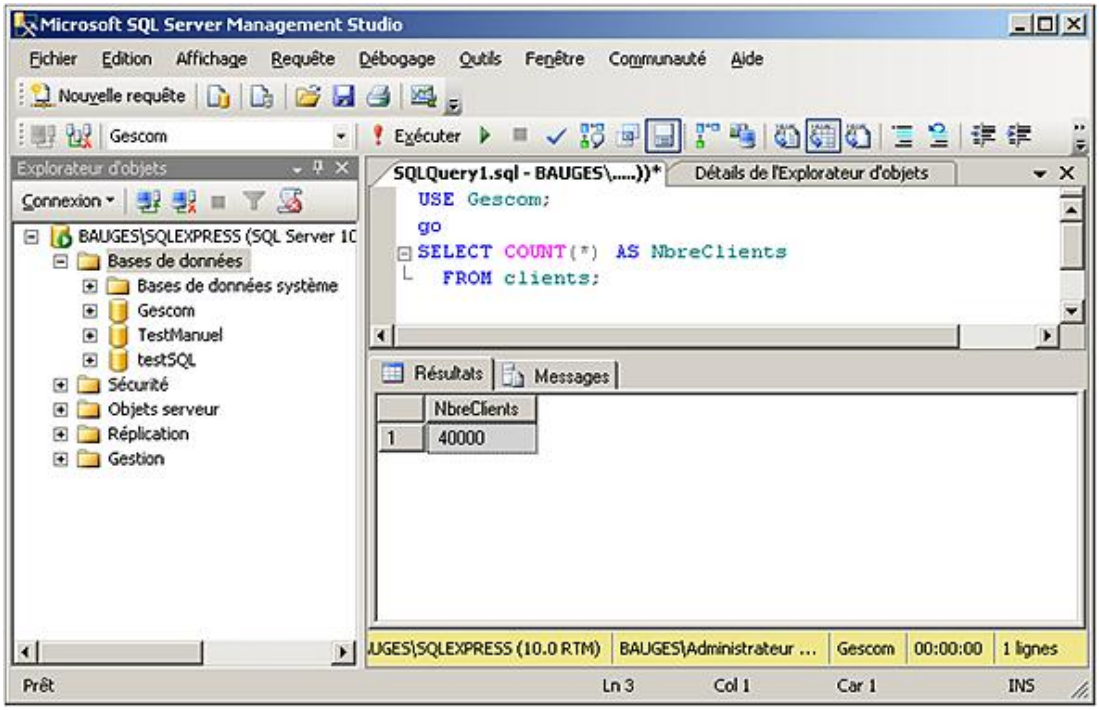
### BEGIN TRAN[SACTION]

Cette instruction permet de démarrer de façon explicite une transaction. En l'absence de cette commande, toute instruction SQL est une transaction implicite qui est validée (COMMIT) aussitôt la modification effectuée sur les données. On parle alors de mode autocommit. Lors du début de la transaction, il est possible de nommer la transaction, mais aussi de marquer le début de la transaction dans le journal de la base de données. Cette marque pourra être exploitée lors d'un processus de restauration des données pour restaurer la transaction.

```
BEGIN { TRAN | TRANSACTION } [nomTransaction]
      [ WITH MARK [ 'description' ] ]
[;]
```

*Exemple*

*Dans l'exemple suivant, une nouvelle transaction est démarrée.*



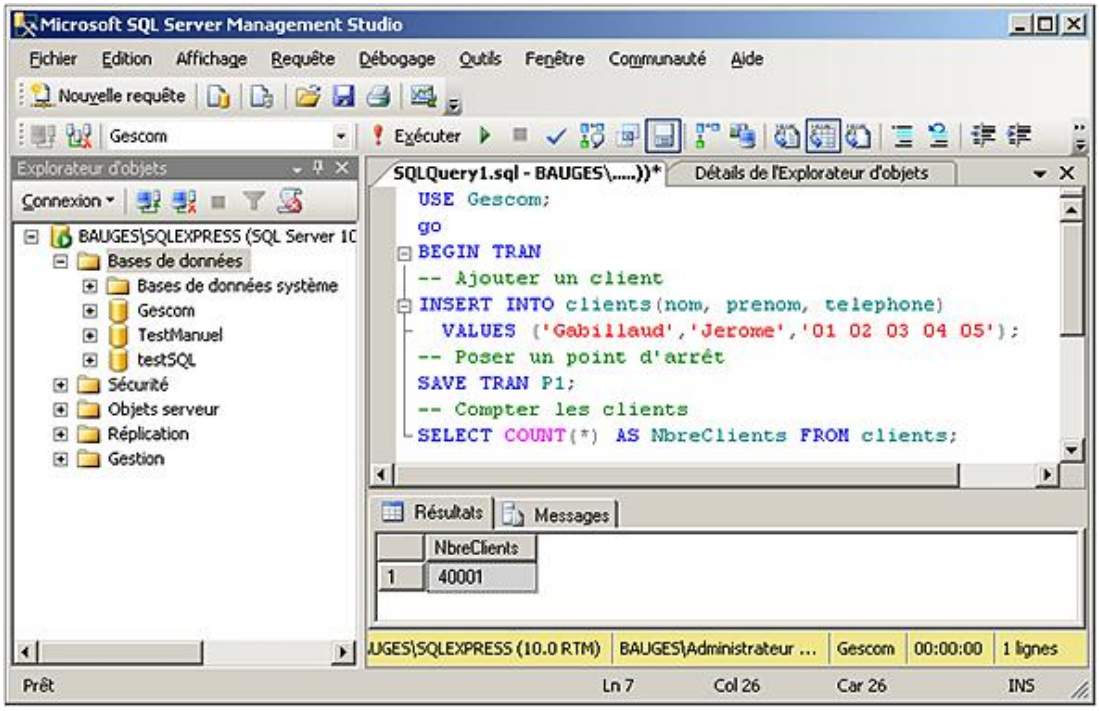

### SAVE TRAN

Cette instruction permet de définir des points d'arrêt et donc donne la possibilité d'annuler une partie de la transaction en cours. Il est possible de définir plusieurs points d'arrêt sur une même transaction. De façon à permettre l'annulation jusqu'à un point d'arrêt précis, ils sont généralement identifiés par un nom.

```
SAVE { TRAN | TRANSACTION } {nomPointArret}[;]
```

*Exemple*

*Dans l'exemple suivant, le point d'arrêt P1 est défini, après l'ajout d'un nouveau client.*



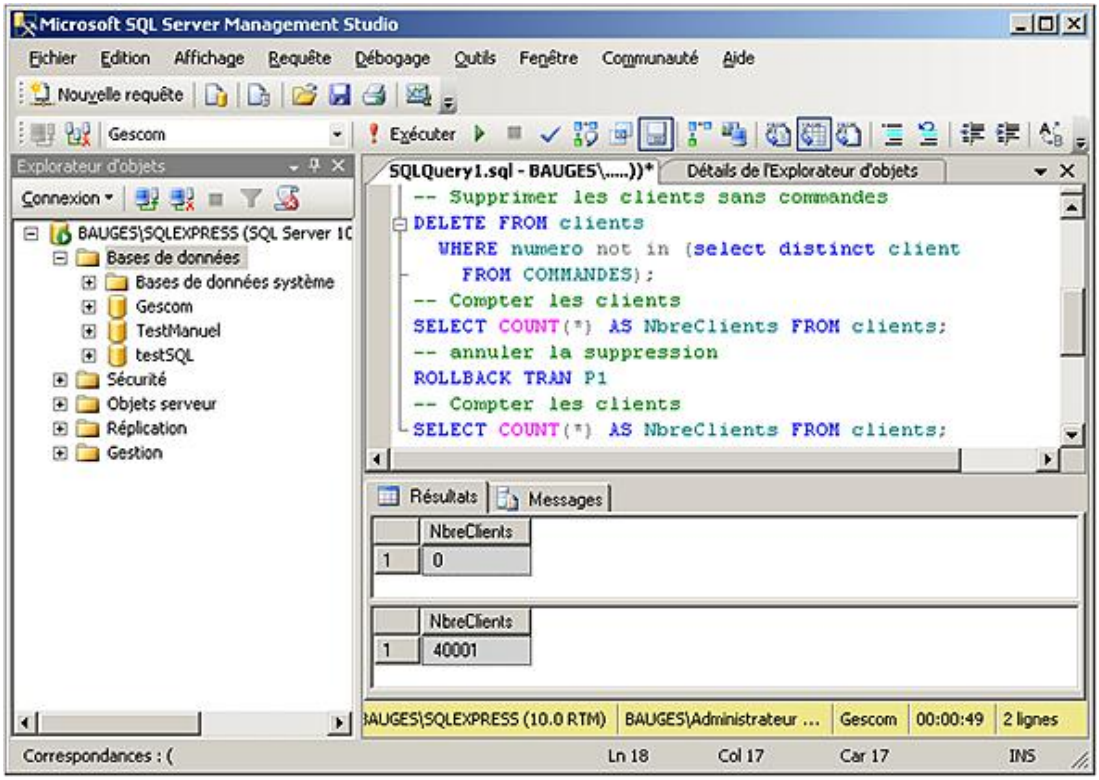

## ROLLBACK TRAN[SACTION]

L'instruction ROLLBACK permet d'annuler une partie ou la totalité de la transaction, c'est-à-dire des modifications intervenues sur les données. L'annulation partielle d'une transaction n'est possible que si des points d'arrêt ont été définis à l'aide de l'instruction SAVE TRAN. Il n'est pas possible d'arrêter l'annulation entre deux points d'arrêt.

```
ROLLBACK { TRAN | TRANSACTION }
         [nomTransaction|nomPointArret][;]
```

*Exemple*

*Dans l'exemple ci-dessous, les clients sans commande sont supprimés, puis une requête compte le nombre de clients définis dans la base. Enfin, la suppression est annulée par l'intermédiaire de l'instruction ROLLBACK qui annule toutes les modifications effectuées sur les données depuis la définition du point d'arrêt P1.*



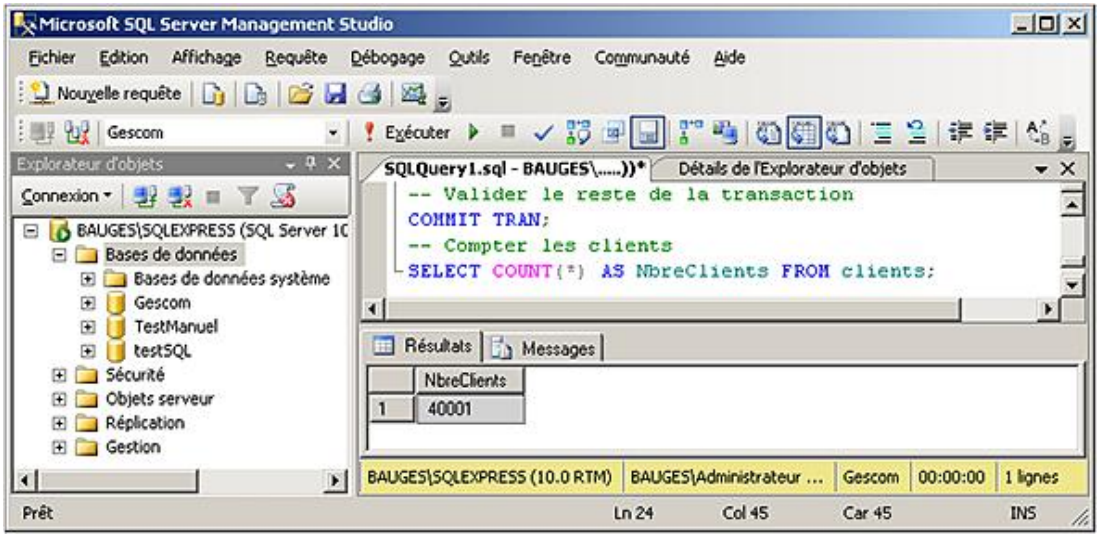

## COMMIT

Cette instruction permet de mettre fin avec succès à une transaction, c'est-à-dire de conserver l'ensemble des modifications effectuées dans la transaction. C'est simplement à l'issue de la transaction que les modifications sont visibles par les autres utilisateurs de la base de données.

```
COMMIT { TRAN | TRANSACTION } [nomTransaction] [;]
```

*Exemple*

*Les modifications sont validées et la transaction prend fin.*




Pour SQL Server, si la transaction n'est pas commencée explicitement par la commande BEGIN TRAN, alors toute instruction SQL constitue une transaction qui est validée automatiquement.

---

Si au cours d'une transaction explicite, le client à l'origine de la transaction rompt brutalement sa connexion avec le serveur, alors la transaction est annulée (ROLLBACK) automatiquement.

---

## 3. Les fichiers journaux

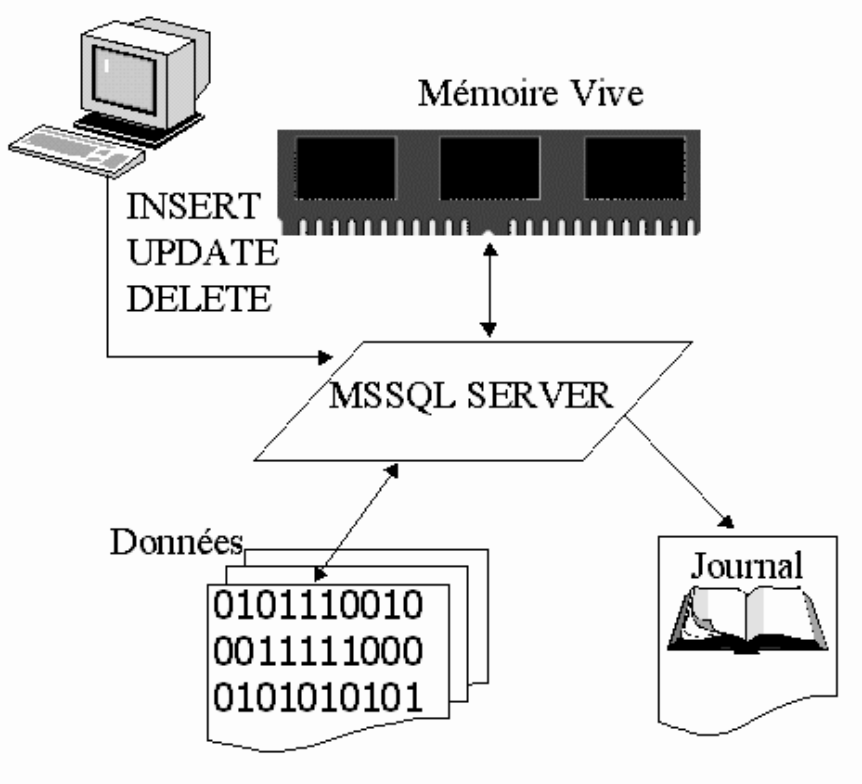
### a. Le rôle

Les fichiers journaux permettent de stocker les images avant et après modification des données contenues dans la base. Seules les opérations du DML (*Langage de Manipulation de Données*), soit les ordres SQL INSERT, UPDATE et DELETE, provoquent une journalisation des opérations qu'elles effectuent sur les données de la base. Les opérations de grande envergure, comme la création d'un index, sont mentionnées dans le journal. Le journal sera utilisé principalement lors des opérations de restauration automatique suite à un arrêt brutal du serveur, ou bien lors des opérations de sauvegardes lorsque ces dernières s'appuient sur le journal. Il sera également utilisé lorsque la base participe à la réplication.

Le but du journal est de permettre au serveur de toujours garantir la cohérence des données, c'est-à-dire que toutes les transactions validées (COMMIT) persistent même s'il arrive un gros problème causant l'arrêt brutal du serveur.

Lors de chaque redémarrage du serveur, SQL Server vérifie que la dernière instruction du journal est un point de synchronisation Si tel n'est pas le cas, toutes les transactions validées (COMMIT) sont rejouées tandis que toutes les transactions non validées sont annulées (ROLLBACK).

### b. Le fonctionnement




*Fonctionnement du journal*

Lorsqu'un ordre SQL est transmis au serveur SQL, ce dernier va chercher à l'exécuter le plus rapidement possible. Après analyse de l'ordre et mise en place du plan d'exécution, si les données concernées par l'ordre ne sont pas déjà présentes en mémoire, alors le moteur SQL va lire les fichiers sur le disque dur afin d'y trouver les informations nécessaires. Une fois présentes en mémoire, les modifications peuvent être apportées aux données. La modification est toujours enregistrée dans le journal avant d'être réellement effectuée sur les données de la base. Un tel journal est appelé journal à écriture anticipée.

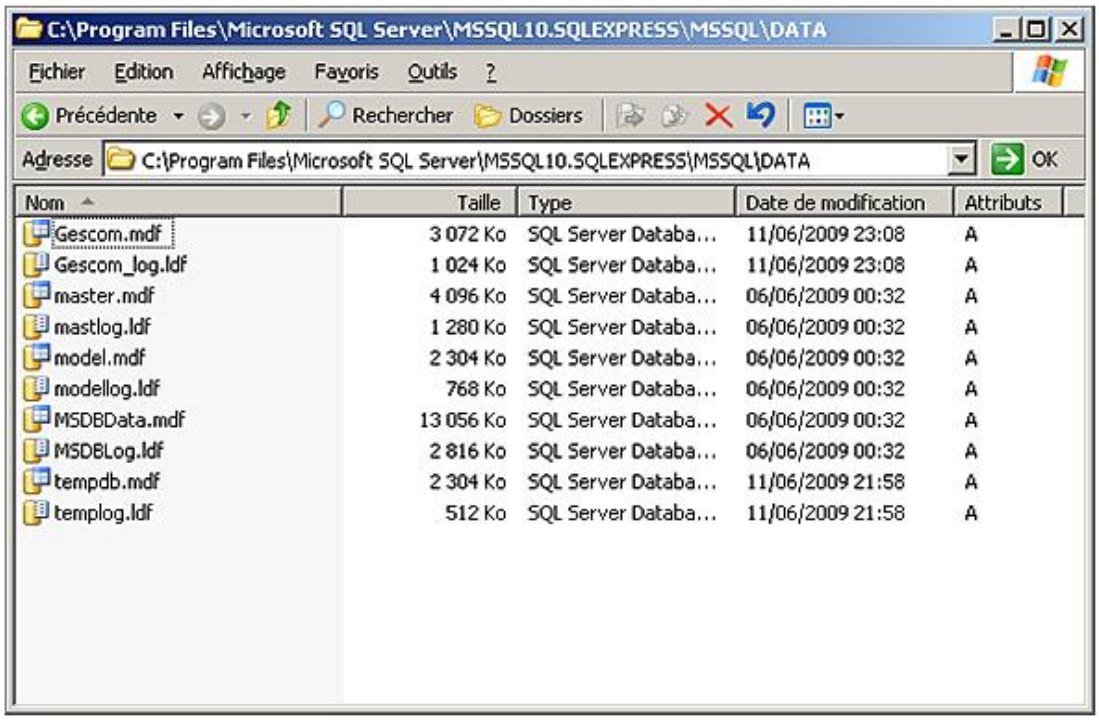
D'une façon logique toutes les informations sont enregistrées les unes à la suite des autres dans le journal. Chaque information est parfaitement identifiée par son LSN (*Log Sequence Number*) ou numéro séquentiel d'enregistrement dans le journal. Chaque enregistrement dans le journal contient également l'identifiant de la transaction à l'origine de la modification des données. Tous les enregistrements d'une même transaction ne sont donc pas enregistrés de façon contiguë.

Les fichiers journaux sont situés, par défaut, dans le répertoire C:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\Data, et portent l'extension *.ldf. Il s'agit d'une extension recommandée qui n'est nullement obligatoire. Le journal peut être constitué d'un ou plusieurs fichiers, la taille de ces derniers pouvant être fixe ou variable automatiquement ou de façon manuelle. La gestion des fichiers sera abordée dans ce chapitre dans la section Créer, gérer et supprimer une base de données.

---

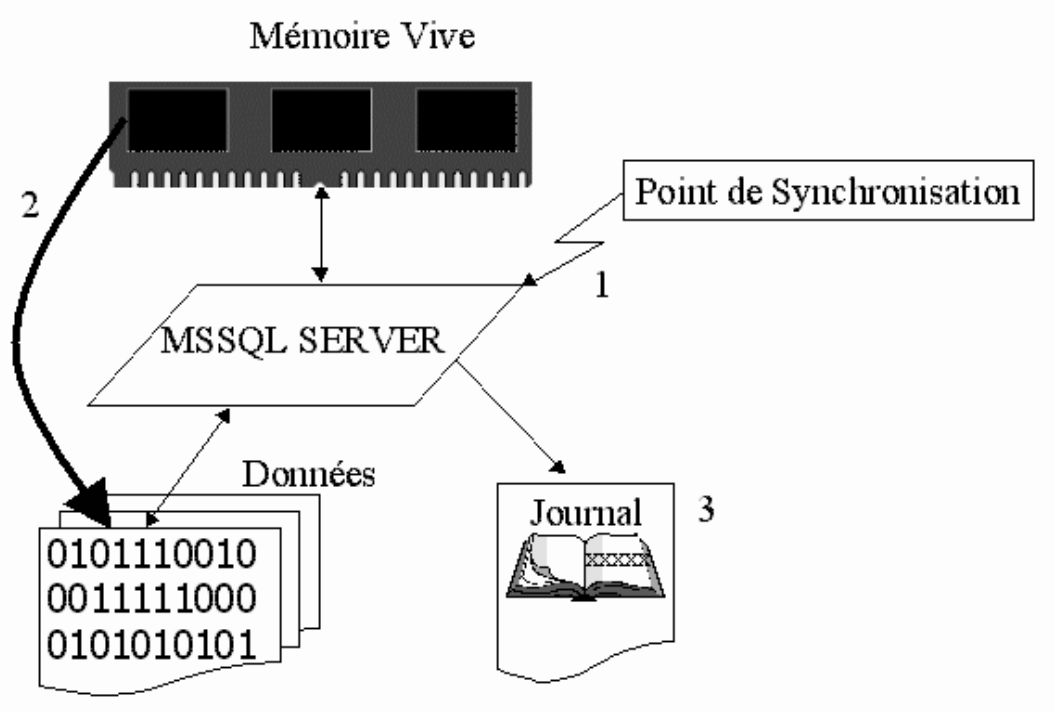
Le journal des transactions peut être constitué de plusieurs fichiers physiques. La gestion du journal est faite de façon indépendante de celle des données. SQL Server gère un cache d'écriture spécifique au journal.

En fonction de l'utilisation faite des informations présentes dans le journal, il est possible de tronquer régulièrement le journal de façon à toujours utiliser les mêmes fichiers physiques, tout en contrôlant l'espace disque occupé.




### c. Les points de synchronisation

Régulièrement, SQL Server va déclencher un point de synchronisation. Il consiste à faire redescendre sur fichier toutes les données stockées en mémoire qui correspondent à des données validées. Les points de synchronisation sont également appelés CHECKPOINT. Le nombre de données touchées entre deux points de synchronisation va déterminer le temps de restauration automatique suite à un arrêt brutal du serveur.



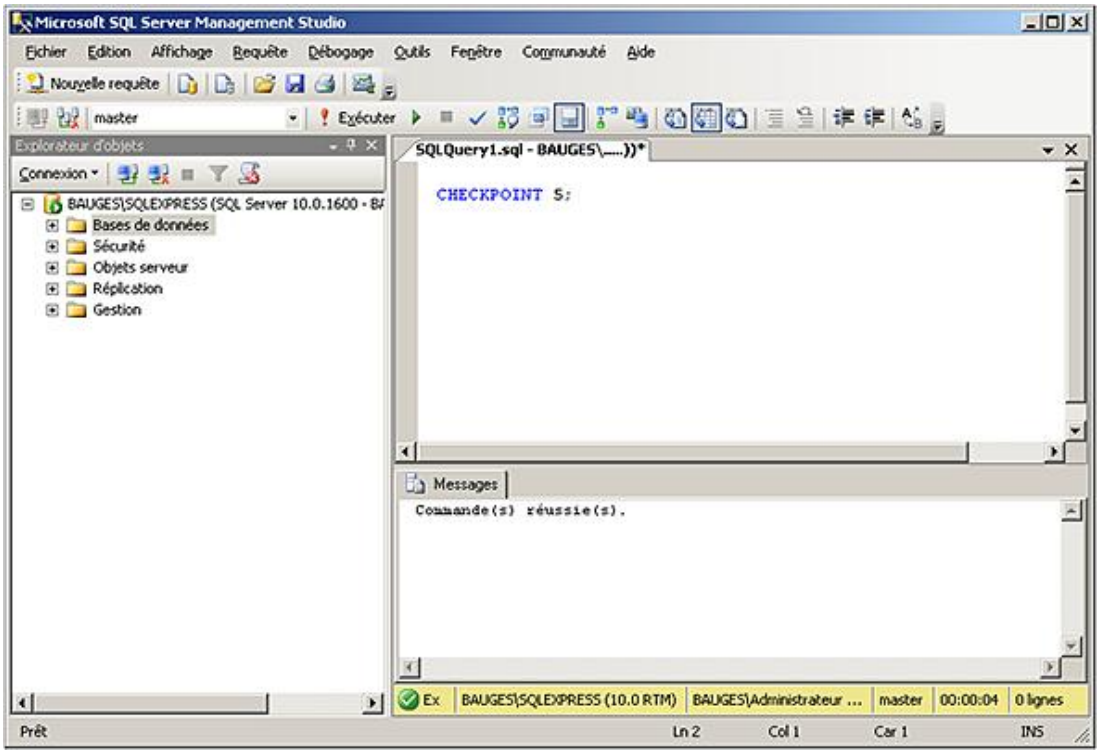

*Principe de fonctionnement d'un point de synchronisation*

SQL Server optimise les points de synchronisation de façon à garantir la meilleure gestion des données sans détériorer les temps de réponse du serveur. Il est toutefois possible d'intervenir sur cette optimisation par l'intermédiaire de l'instruction CHECKPOINT.

```
CHECKPOINT [tempsRealisation]
```

`tempsRealisation`

Permet de préciser le temps accordé en secondes pour terminer le point de synchronisation. C'est SQL Server qui se charge de déclencher le point de synchronisation de façon à avoir terminé dans les temps.



Seul un arrêt de l'instance par l'intermédiaire de l'instruction Transact SQL SHUTDOWNWITH NOWAIT permet d'arrêter l'instance sans déclenchement d'un point de synchronisation.

---

# 4. Les fichiers de données

## a. Leur rôle

Chaque base de données possède au moins un fichier de données. Ce fichier va contenir l'ensemble des données stockées dans la base. Chaque fichier de données ne peut contenir que des données en provenance d'une seule base, il y a donc spécialisation des fichiers par rapport à la base de données.

## b. La structure des fichiers de données

Les fichiers de données sont structurés pour répondre de manière optimum à toutes les sollicitations de la part du moteur et surtout pour être capables de stocker plus de données en optimisant l'espace disque utilisé.

Pour optimiser l'espace dont il dispose, le serveur va formater les fichiers de données de façon à maîtriser leur structure.

Le travail réalisé par SQL Server sur ces fichiers de données est semblable au travail fait par le système d'exploitation sur les disques disponibles sur la machine. Il est tout à fait admis que le système d'exploitation formate l'espace disque dont il dispose afin de le diviser en blocs. Par la suite ce sont ces blocs qui vont être accordés aux fichiers. SQL Server réalise le même type de travail sur les fichiers de données, puis accorde l'espace disponible aux différentes tables et index.

### Les pages

Avant de pouvoir travailler avec un fichier de données, SQL Server va structurer le fichier en le découpant en bloc ou page de 8 Ko. La taille de 8 Ko est fixée par SQL Server, de façon à réduire les pertes d'espace tout en simplifiant les opérations d'allocation, mais aussi de lecture et d'écriture. La page représente l'unité de travail de SQL Server. C'est toujours une page entière de données qui est remontée en mémoire et c'est toujours une page qui est écrite sur le disque. Il n'est pas possible de remonter en mémoire cache, une partie d'une page. Bien entendu, une page peut contenir plusieurs lignes d'une table ou bien plusieurs entrées d'index et toutes les informations sont remontées en une seule lecture disque.

Cette taille de 8 Ko permet également de gérer des bases de données de plus grande taille avec un nombre de blocs moindre.

Enfin, pour éviter la fragmentation des lignes de données sur plusieurs blocs, une ligne doit toujours être entièrement contenue (hors type text et image) dans un bloc. La taille maximale d'une ligne est donc de 8060 octets. Cette valeur de 8060 octets est légèrement inférieure à 8 Ko car SQL Server réserve quelques octets pour gérer l'en-tête de la page afin d'en maîtriser la gestion.

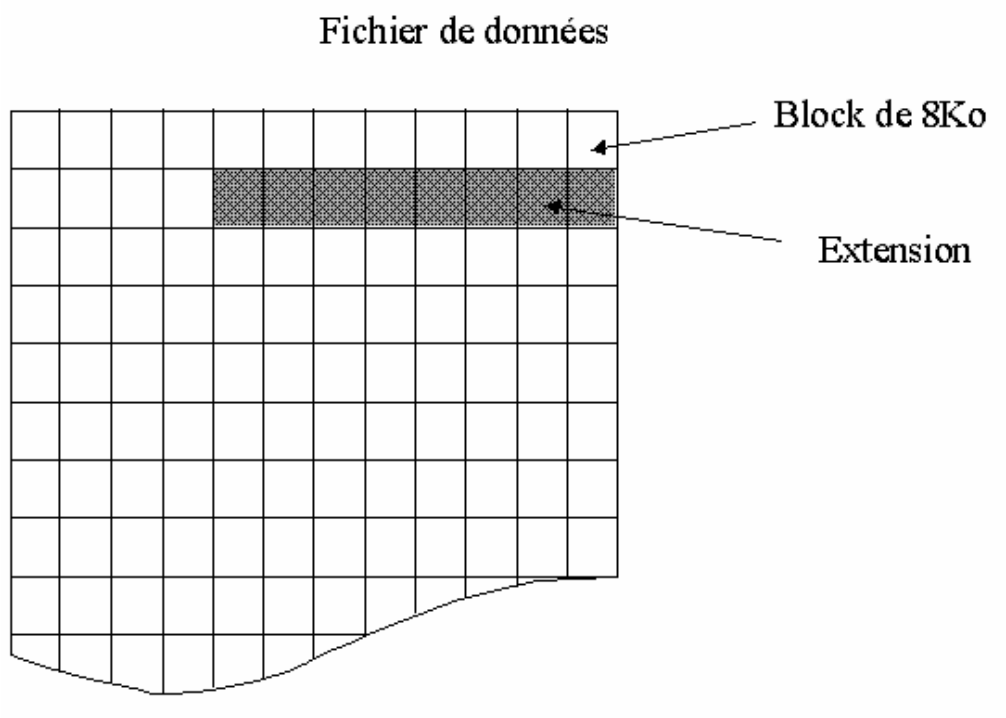
Si plusieurs lignes sont stockées dans la même page, elles le sont de façon séquentielle. En cas de changement de taille des lignes, SQL Server gère dynamiquement le déplacement des lignes dans la page ou bien la mise en place d'un pointeur vers une autre page afin de lire la fin de la ligne de données.

Étant donné que la page est l'unité de travail de SQL Server, chaque page contient un type bien précis de données. Il est possible de distinguer les types de page suivants :

- données : ces pages contiennent des informations au format numérique, texte ou bien date.
- Texte/image : ces pages contiennent soit des textes volumineux, soit des objets au format binaire.
- Index : ces pages contiennent les entrées des index.
- GAM/SGAM : ou *Global Allocation Map* et *Shared Global Allocation Map*. Ces pages contiennent des informations relatives à l'allocation des extensions.
- PFS : ou *Page Free Space*. Ces pages contiennent les informations relatives à l'allocation des pages et l'espace disponible sur ces pages.
- IAM : ou *Index Allocation Map*. Ces pages contiennent les informations relatives à l'utilisation de pages par les tables et les index.

- BCM : ou *Bulk Change Map*. Ces pages contiennent la liste des extensions modifiées par des opérations de copie en bloc depuis la dernière sauvegarde du journal.

- DCM : ou *Différentiel Change Map*. Ces pages contiennent la liste de toutes les extensions modifiées depuis la dernière sauvegarde de la base.




*Le découpage en bloc des fichiers de données*

## Les extensions

Les extensions sont des regroupements logiques de 8 blocs contigus, elles ont donc une taille de 64 Ko (8 x 8 Ko). Le rôle des extensions est d'éviter une trop grande dispersion des données pour un même objet au sein des fichiers de données.

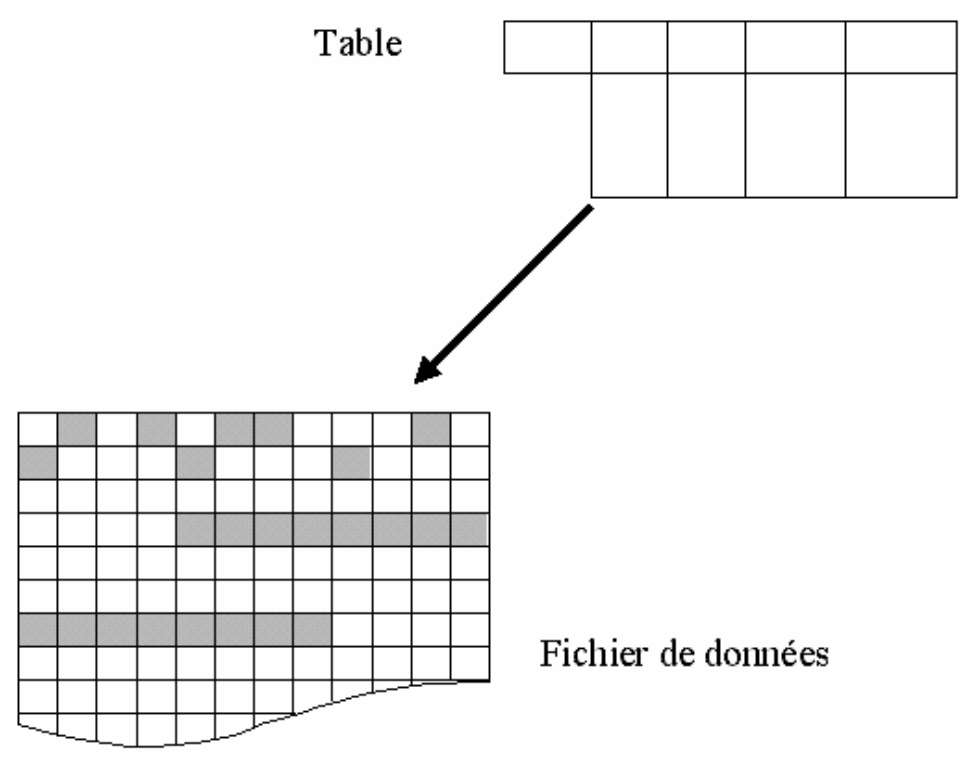
Il existe deux types d'extension : les extensions mixtes et les extensions propres ou uniformes. Ces deux types d'extension permettent de limiter au mieux la consommation d'espace disque par une table en fonction du volume de données à stocker.

### Les extensions mixtes

Au départ, les extensions sont communes à plusieurs tables ou plusieurs index. Lorsqu'un des objets définis sur l'extension demande la place, SQL Server octroie de la place bloc par bloc tant que l'objet n'utilise pas 8 blocs. Dès qu'un objet franchit cette barrière, SQL Server lui octroie de la place extension par extension. L'avantage de cette méthode est d'éviter de perdre de la place inutilement avec les tables ou index contenant peu de données.

### Les extensions uniformes

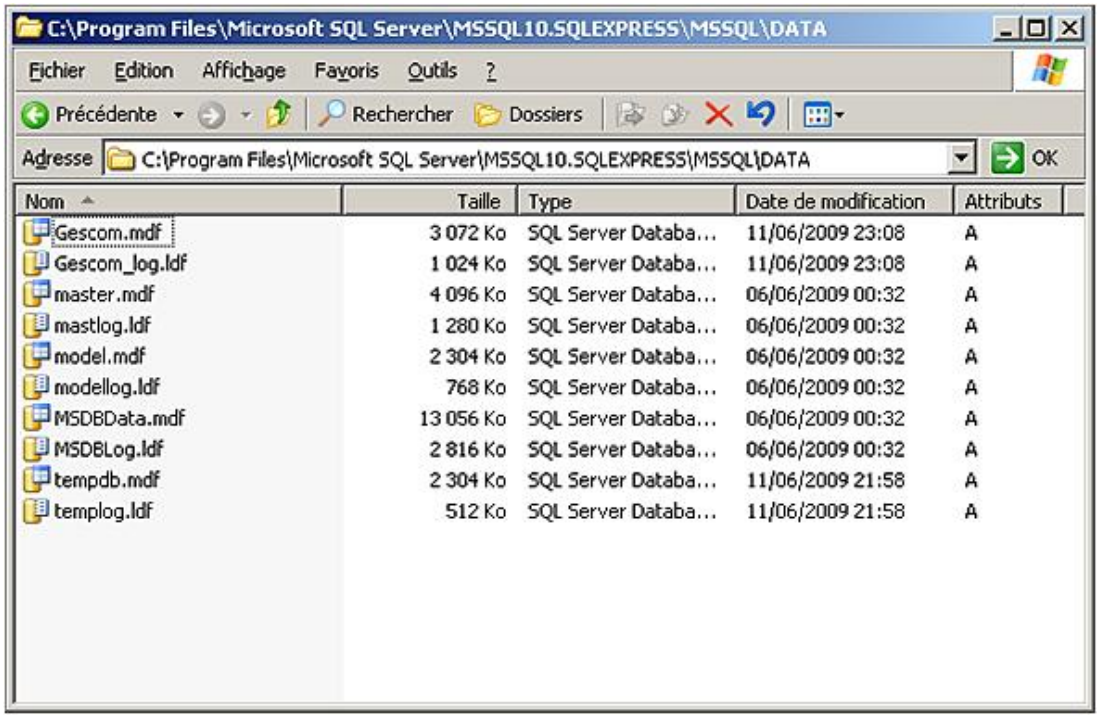
Si un objet occupe plus de 64 Ko d'information, alors la place lui sera accordée extension par extension. Chacune des extensions étant spécialisée pour un objet, les lectures disque seront d'autant plus rapides car toutes les données sont stockées de façon contiguë par paquets de 64 Ko.




*L'allocation des extensions*

### c. Le fonctionnement

Chaque base possède au moins un fichier de données, ce fichier est spécifique à une base. Son chemin par défaut est C:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\Data. Le premier fichier de données d'une base porte l'extension mdf, tous les suivants portent l'extension ndf. Ces extensions sont recommandées mais ne sont pas obligatoires.



# Créer, gérer et supprimer une base de données

Une base de données gère un ensemble de tables système et des tables utilisateurs. Les informations contenues dans ces tables système représentent, entre autres, la définition des vues, des index, des procédures, des fonctions, des utilisateurs et des privilèges. Les tables utilisateurs vont contenir les informations saisies par les utilisateurs.

Une instance SQL Server ne peut pas contenir plus de 32767 bases de données.

## 1. Créer une base de données

La création d'une base de données est une étape ponctuelle, réalisée par un administrateur SQL Server. Avant de tenter de créer une base de données, il est important de définir un certain nombre d'éléments de façon précise :

- le nom de la base de données qui doit être unique sur le serveur SQL,
- la taille de la base de données,
- les fichiers utilisés pour le stockage des données.

Pour créer une nouvelle base de données, SQL Server s'appuie sur la base Model. Cette base Model contient tous les éléments qui vont être définis dans les bases utilisateurs. Par défaut, cette base Model contient les tables système. Il est cependant tout à fait possible d'ajouter des éléments dans cette base. Toutes les bases utilisateurs créées par la suite disposeront de ces éléments supplémentaires.

Une base peut être créée de deux façons différentes :

- par l'intermédiaire de l'instruction Transact SQL CREATE DATABASE ;
- par l'intermédiaire de SQL Server Management Studio.

Une base de données est toujours composée au minimum d'un fichier de données principal (extension mdf) et d'un fichier journal (extension ldf). Des fichiers de données secondaires (ndf) peuvent être définis lors de la création de la base ou bien ultérieurement.

Cette opération de création de base de données affecte la base Master. Une sauvegarde de cette base système s'avère donc nécessaire pour être en mesure de travailler avec la nouvelle base suite à une restauration.

Les informations concernant les fichiers de données sont enregistrées dans la base Master ainsi que dans le fichier primaire de la base de données.

### a. La syntaxe Transact SQL

```
CREATE DATABASE nomBaseDeDonnées
[ ON
  [PRIMARY] [ <spécificationFichier> [,n]]
  [LOG ON < spécificationFichier > [,n]]
]
[ COLLATE classement ]
[;]
```

Avec pour spécificationFichier les éléments de syntaxe suivants :

```
(NAME = nomLogique,
FILENAME = 'cheminEtNomFichier'
[,SIZE = taille [KB|MB|GB|TB]]
[,MAXSIZE={tailleMaximum[KB|MB|GB|TB]|UNLIMITED}]
[,FILEGROWTH = pasIncrement [KB|MB|GB|TB|%]]
) [,n]

PRIMARY
```

Permet de préciser le premier groupe de fichiers de la base. Ce groupe est obligatoire, car l'ensemble des tables système est obligatoirement créé dans le groupe primary. Si le mot clé PRIMARY est omis, le premier fichier de données précisé dans la commande CREATE DATABASE correspond obligatoirement au fichier primaire, il porte normalement l'extension mdf.

NAME

Cet attribut, obligatoire, permet de préciser le nom logique du fichier. Ce nom sera utilisé pour faire des manipulations sur le fichier via des commandes Transact SQL ; on pense notamment aux commandes DBCC ou à la gestion de la taille des fichiers.

FILENAME

Spécification du nom et emplacement physique du fichier sur le disque dur. Le fichier de données est toujours créé sur un disque local du serveur.

SIZE

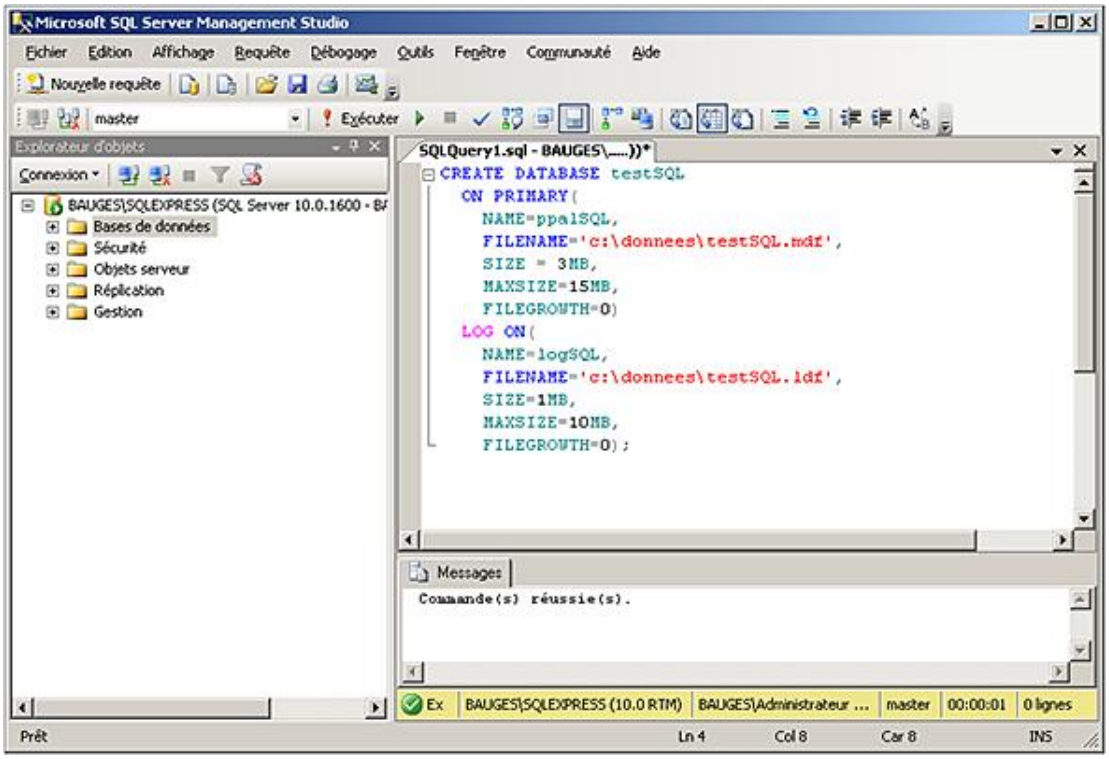
Attribut optionnel qui permet de préciser la taille du fichier de données. La taille par défaut est de 1 Mo et la taille minimale d'un fichier est 512 Ko aussi bien pour le journal que pour les fichiers de données. La taille peut être précisée en kilo-octets (Kb) ou en mégaoctets (Mb, par défaut). La taille du premier fichier de données doit être au moins égale à celle de la base `Model`.

MAXSIZE

Cet attribut optionnel, permet de préciser la taille maximale en Mégaoctets (par défaut) ou en kilo-octets que peut prendre le fichier. Si rien n'est précisé, le fichier grossira jusqu'à saturation du disque dur.

FILEGROWTH

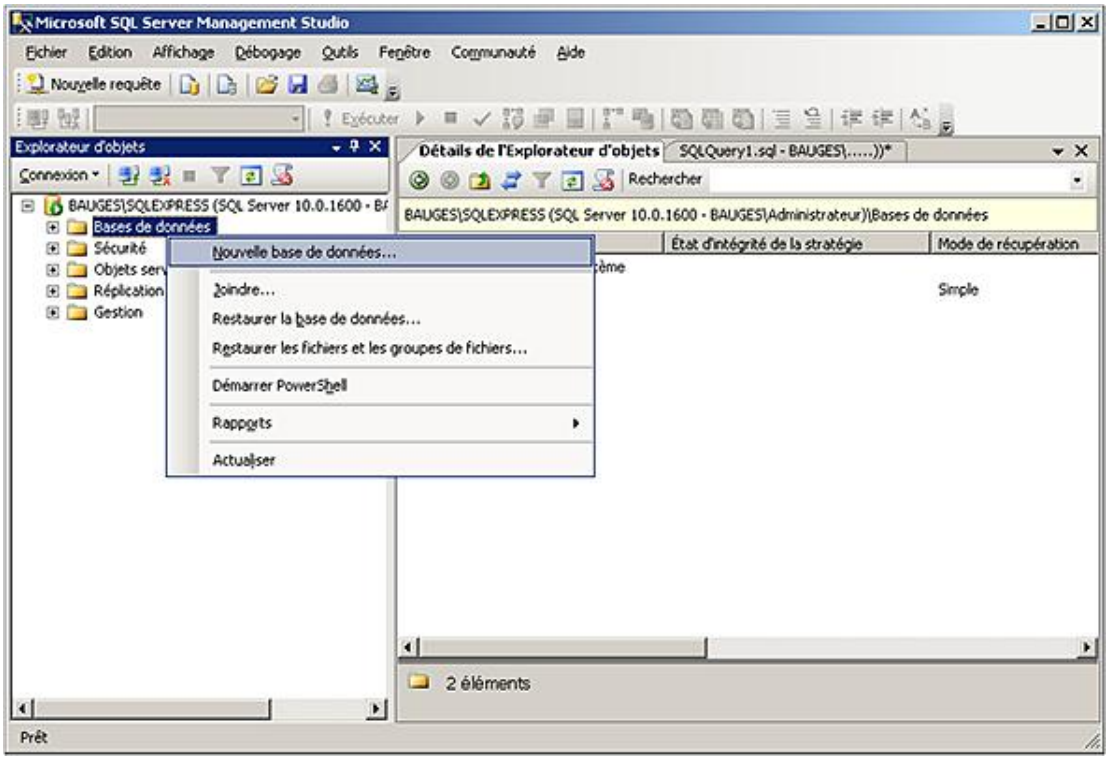
Il s'agit de préciser le facteur d'extension du fichier. La taille de chacune des extensions peut correspondre à un pourcentage (%), à une taille en Mégaoctets (par défaut) ou en Kilo octets. Si on précise la valeur zéro, le facteur d'extension automatique du fichier est nul et donc la taille du fichier ne change pas automatiquement. Si le critère FILEGROWTH est omis, la valeur par défaut est de 1 Mo pour les fichiers de données et de 10 % pour les fichier journaux. La taille des extensions est toujours arrondie au multiple de 64 Ko (8 blocs) le plus proche.



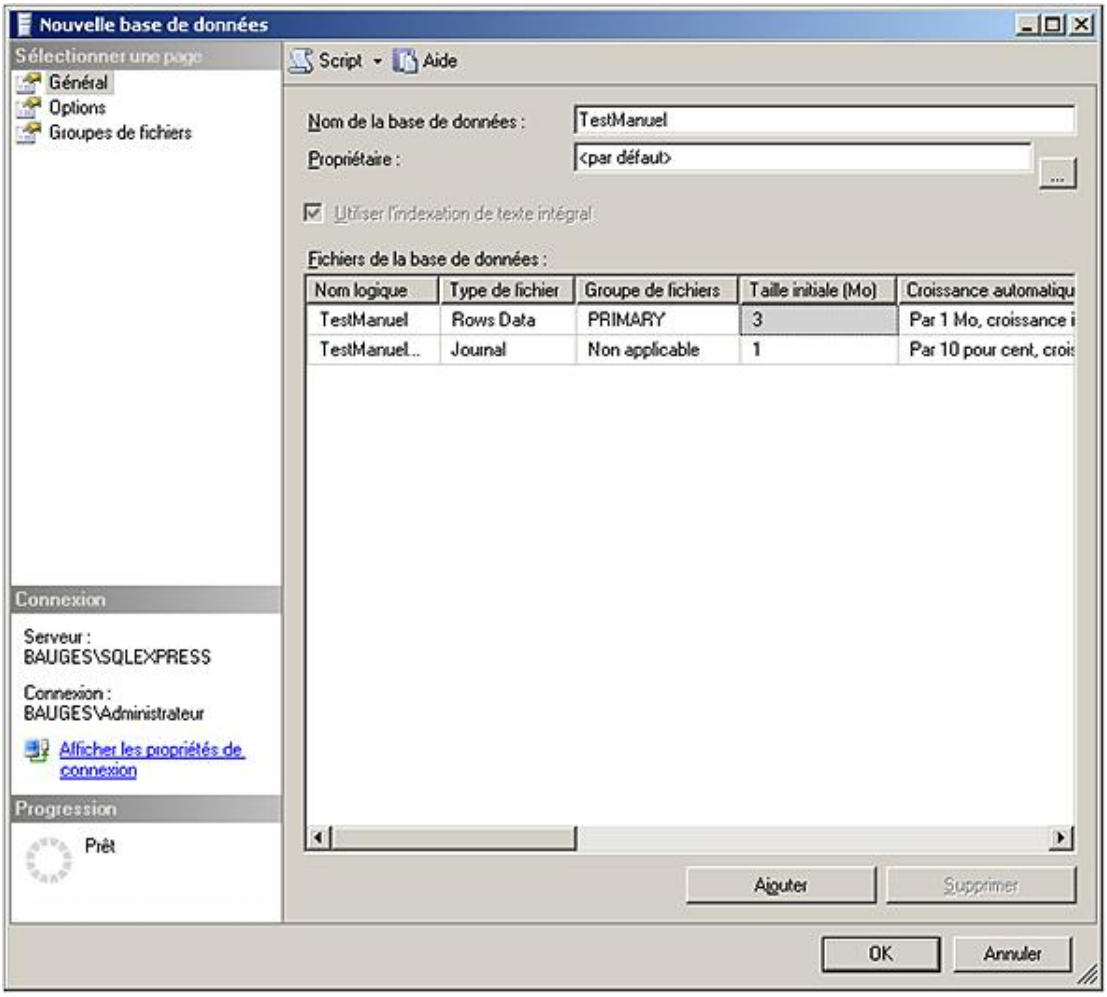

Toutes les options de l'instruction CREATE DATABASE ne sont pas exposées ici. Seules les options les plus couramment utilisées lors de la création d'une nouvelle base le sont.

## b. Utilisation de SQL Server Management Studio

Il est également possible de créer une base de données de façon graphique depuis SQL Server Management Studio. Il faut alors sélectionner **Nouvelle base de données** depuis le menu contextuel associé au nœud **Bases de données** depuis l'explorateur d'objets, comme illustré ci-dessous.




SQL Server Management Studio présente alors la boîte de dialogue des propriétés de la base en mode création. Après avoir complété les options comme le nom de la base et le nom et l'emplacement des fichiers, il est possible de demander la création de la base.




Cette boîte de dialogue permet à un utilisateur averti de créer rapidement et simplement une base de données, tout en conservant la maîtrise de tous les paramètres.

Il est possible de créer des fichiers sur des partitions brutes. Cependant, cette technique n'offre qu'un faible gain de performance par rapport à la création de fichiers sur des partitions formatées NTFS. De plus, les fichiers créés sur des partitions brutes ne peuvent pas être manipulés par le système de fichiers (notamment dans le cadre des sauvegardes base arrêtée) et il ne peut y avoir, bien sûr, qu'un seul fichier par partition.

Les fichiers de base de données ne doivent pas être créés sur des partitions compressées. Dans certains cas extrêmes, il est possible de placer les fichiers secondaires sur des partitions compressées. Les fichiers doivent alors être en lecture seule. Par contre, les fichiers journaux ne doivent en aucun cas être définis sur une partition compressée.

## 2. Gérer une base de données

Lors de la gestion d'une base de données, plusieurs critères sont à prendre en compte. Dans cette section, la gestion de l'espace utilisé par les fichiers physiques qui constituent la base de données est abordée. Les points principaux qui concernent la gestion des fichiers sont : l'accroissement dynamique ou manuel des fichiers, l'ajout de nouveaux fichiers, la réduction de la taille des fichiers.

### a. Augmenter l'espace disque disponible pour une base de données

Les fichiers de données et les fichiers journaux stockent de l'information. Comme la base contient normalement de plus en plus d'informations, à un moment donné ces fichiers seront complets. Il se posera alors le problème de savoir où prendre la place.

Les différentes méthodes exposées ci-dessous pour augmenter l'espace de stockage dont dispose la base sont complémentaires car chaque méthode possède ses avantages et ses inconvénients.

#### Fichier à accroissement dynamique

Lors de la création de la base, il est possible de fixer un certain nombre de critères concernant la taille maximale de

fichiers (MAXSIZE) et le taux d'accroissement (FILEGROWTH). Si ces options sont omises la taille maximale est infinie et le taux d'accroissement est de 10 % pour les journaux et 1 Mo pour les fichiers de données.

En utilisant des fichiers à croissance dynamique, le serveur ne sera jamais bloqué par la taille du fichier sauf saturation du disque ou taille maximale atteinte.

Le facteur d'accroissement permet de fixer la taille de tous les ajouts. Cette taille doit être fixée de façon optimale, en prenant en compte le fait qu'un facteur d'accroissement trop petit introduit beaucoup de fragmentations physiques du fichier et les demandes d'extension du fichier sont fréquentes, tandis qu'un facteur d'accroissement trop grand nécessite une charge de travail importante de la part du serveur lorsque celui-ci met en place la structure de blocs de 8 Ko à l'intérieur du fichier.

Si le taux d'accroissement est fixé à zéro, le fichier ne pourra pas grandir dynamiquement.

L'accroissement du fichier possède toujours une taille qui est multiple de 64 Ko (taille d'une extension).

Si le paramétrage des fichiers permet de gérer automatiquement la croissance des fichiers, cette solution présente tout de même le désavantage du fait qu'il n'est pas possible de maîtriser quand cet accroissement aura lieu. Si cette opération intervient en pleine charge de travail, elle risque de ralentir le serveur.

Par contre, si le paramétrage de l'accroissement automatique des fichiers est réalisé, cela permet en cas de problème, que les fichiers adaptent leur taille en fonction du volume de données, sans bloquer les utilisateurs.

### Fichier à accroissement manuel

La croissance manuelle des fichiers permet de maîtriser le moment où le fichier va grossir et donc le moment où le serveur va subir une charge de travail supplémentaire pour mettre en forme le fichier s'il s'agit d'un fichier de données.

### Ajout de fichiers

Pour permettre à une base de données d'obtenir plus de place, il est enfin possible de rajouter des fichiers. Cette solution présente le double avantage de maîtriser l'instant où le serveur va subir une surcharge de travail, ainsi que de ne pas fragmenter physiquement les fichiers surtout si ces derniers sont stockés sur une partition NTFS. Cependant cette solution nécessite que l'administrateur observe de près l'utilisation des fichiers journaux et de données afin de toujours intervenir avant que le système ne se bloque pour un manque d'espace disque.

### Le journal des transactions

Le journal des transactions est composé de un à plusieurs journaux. Afin que le serveur fonctionne correctement, il est indispensable que le journal ne soit jamais saturé. Le journal est vidé lors des sauvegardes et éventuellement à chaque point de synchronisation si la base est configurée en mode de restauration simple.

Pour assurer une place suffisante au journal, la solution la plus facile consiste à positionner le ou les fichiers qui le composent avec une croissance automatique.

### Modifier un fichier en Transact SQL

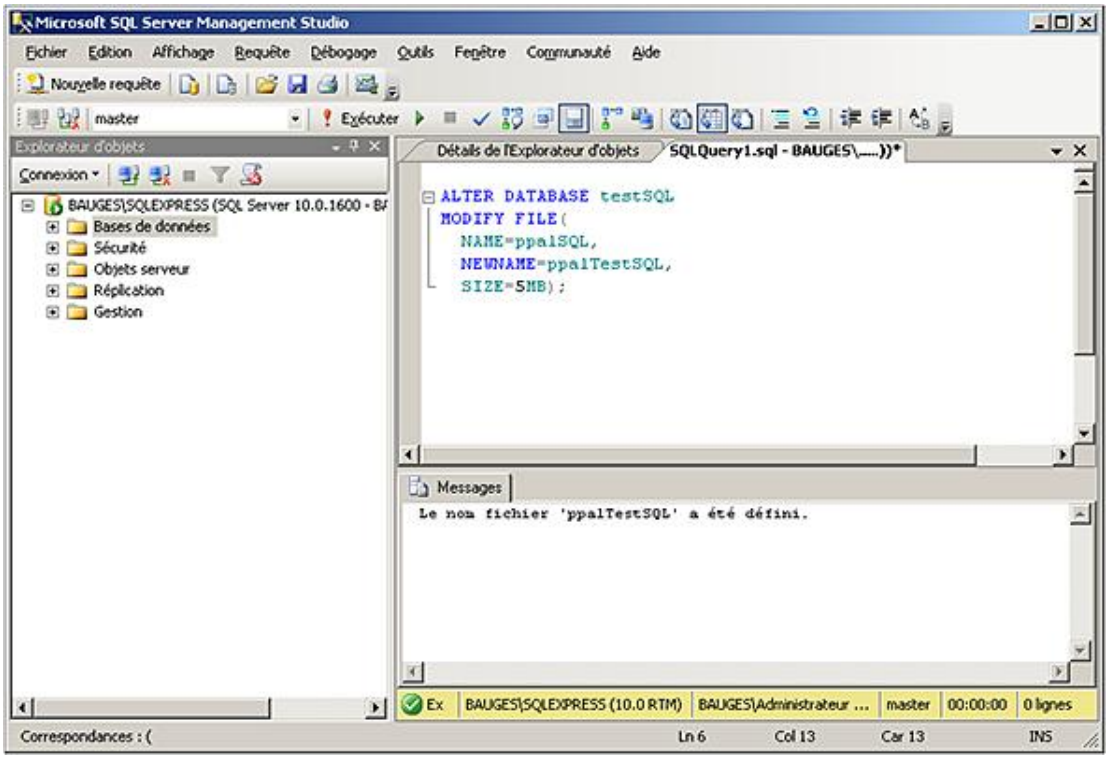
C'est l'instruction ALTER DATABASE qui permet d'effectuer toutes les opérations relatives aux manipulations des fichiers de base de données, aussi bien pour les fichiers de données que les fichiers du journal de transaction.

Toutes les caractéristiques des fichiers peuvent être modifiées, mais les modifications apportées doivent suivre certaines règles comme, par exemple, la nouvelle taille du fichier qui doit être supérieure à la taille initiale.

```
ALTER DATABASE nomBaseDeDonnées
MODIFY FILE (spécificationFichier)[;]
```

Avec pour `spécificationFichier` les éléments de syntaxe suivants :

```
(NAME = nomLogique,
NEWNAME = nouveauNomlogique,
FILENAME = 'cheminEtNomFichier'
[,SIZE = taille [KB|MB|GB|TB]]
[,MAXSIZE={tailleMaximum[KB|MB|GB|TB]|UNLIMITED}]
[,FILEGROWTH = pasIncrement [KB|MB|GB|TB|%]]
)
```

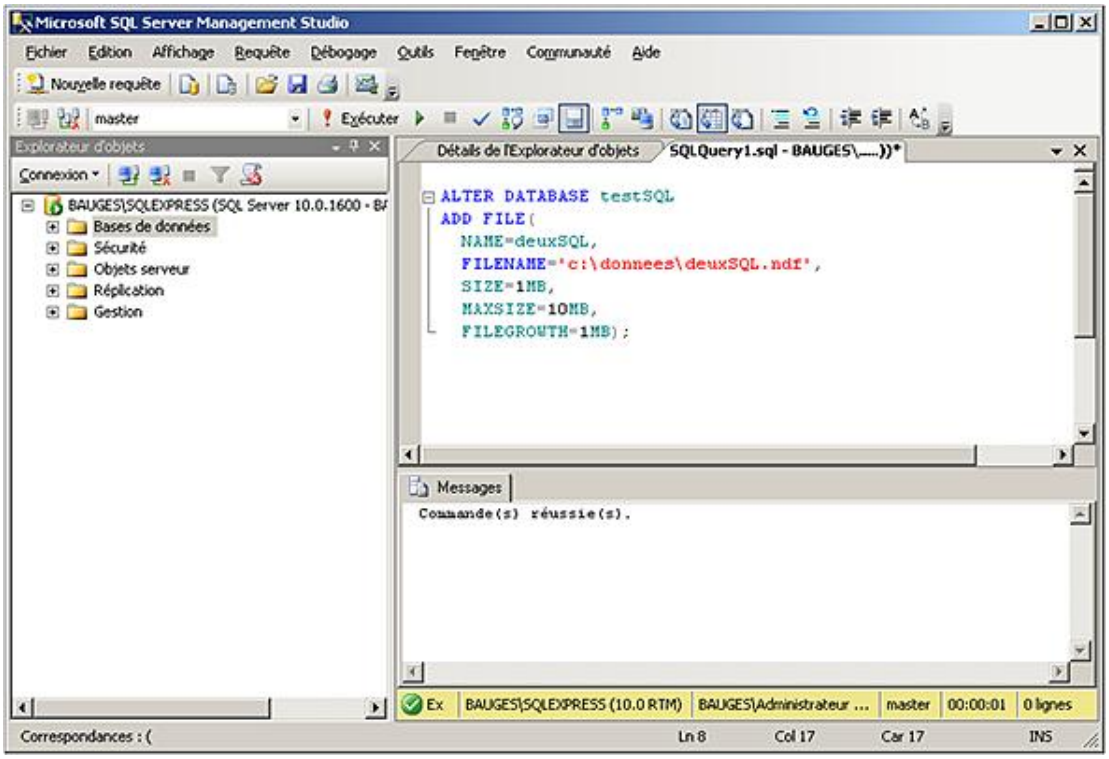



## Ajouter un fichier en Transact SQL

C'est la commande ALTER DATABASE associée à l'option ADD FILE qui va permettre de rajouter un fichier de données ou un fichier journal.

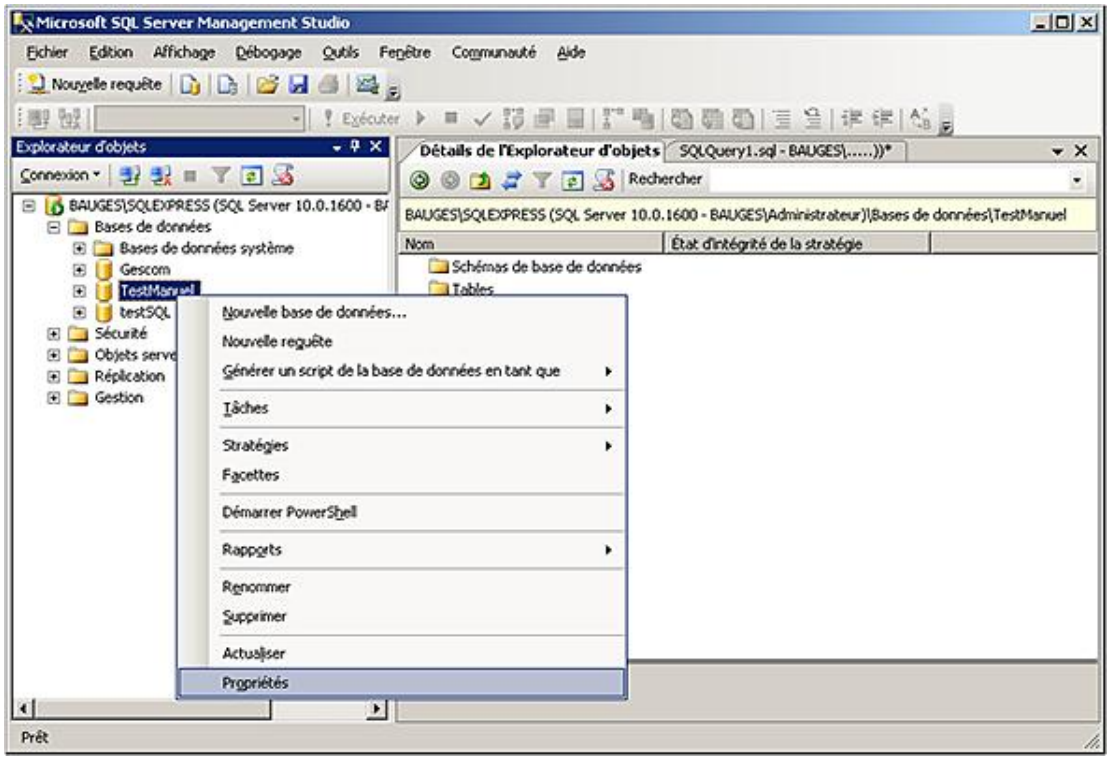
*Syntaxe :*

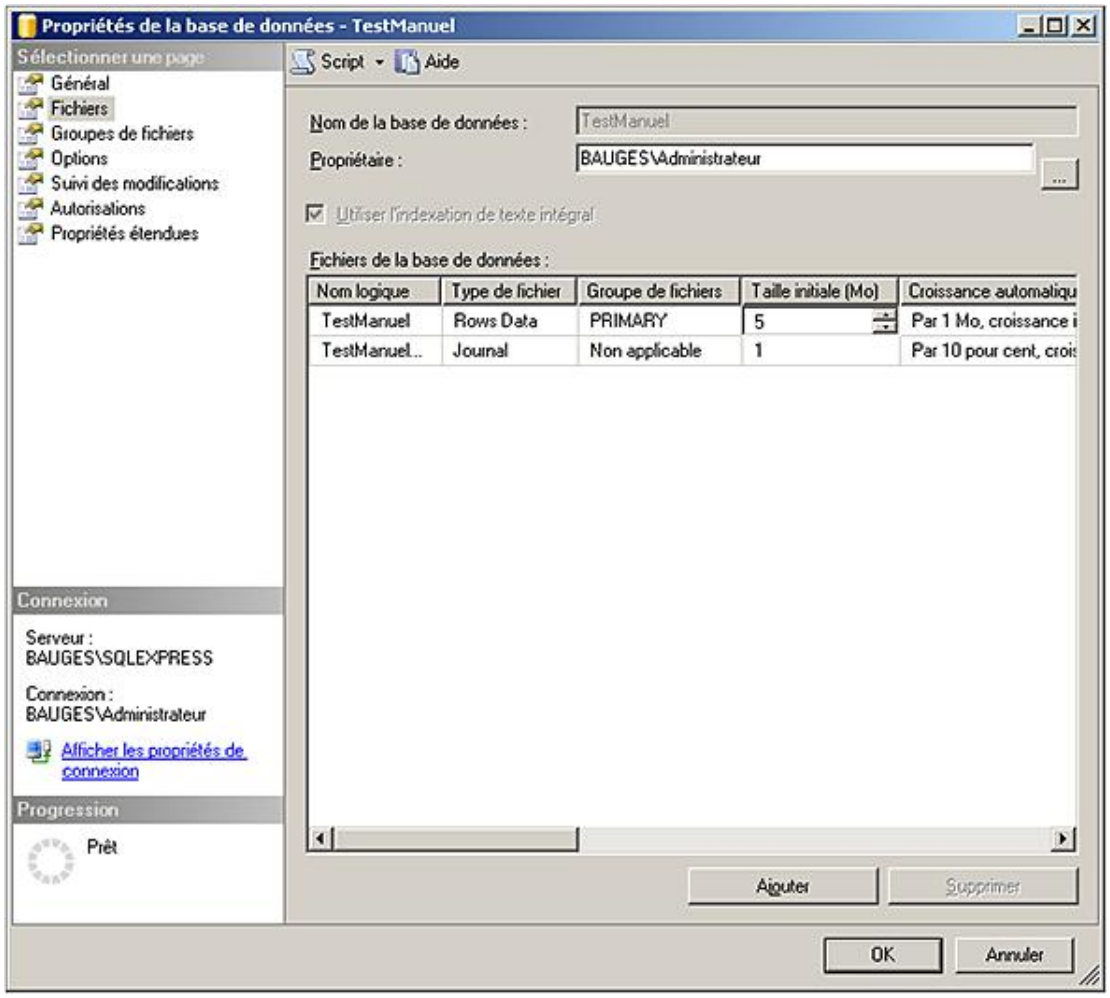
```
ALTER DATABASE nomBaseDeDonnées
ADD FILE spécificationFichier[;]
```




## Modifier et ajouter des fichiers depuis SQL Server Management Studio

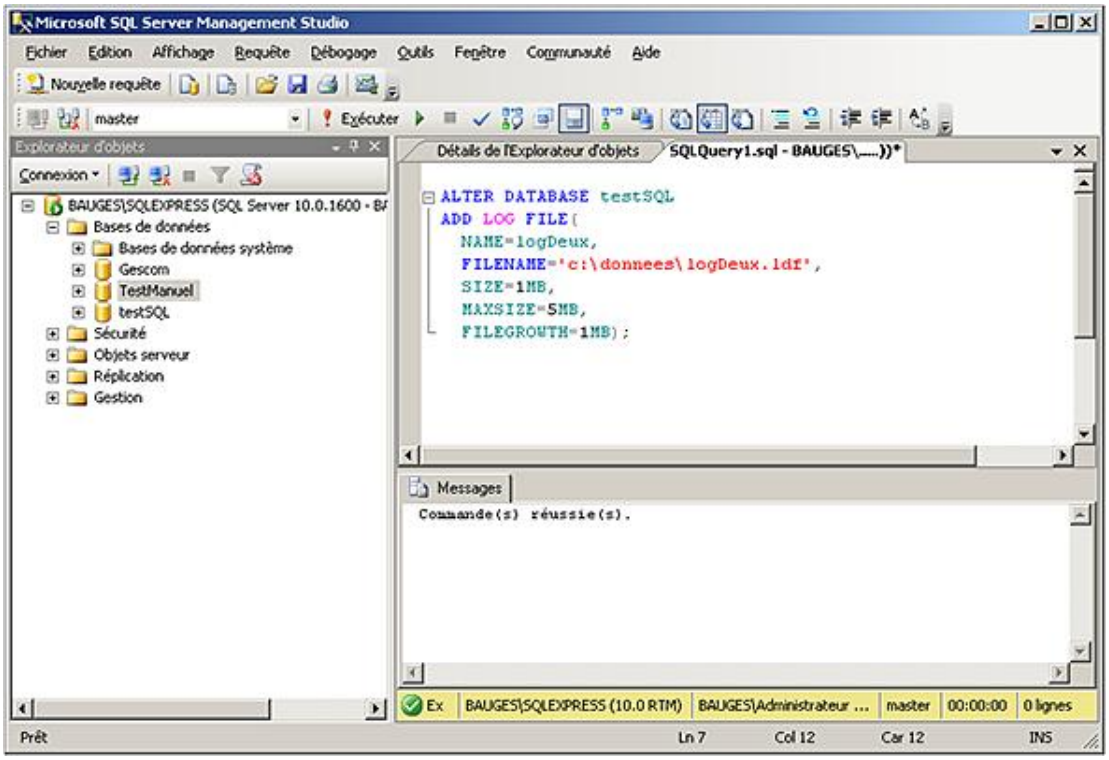
C’est par l’intermédiaire de la boîte de dialogue qui indique les propriétés de la base de données, qu’il est possible de gérer les opérations sur la taille initiale des fichiers ainsi que leur taille maximale.




## Les fichiers du journal

Comme pour les fichiers de données, il est possible de redimensionner le fichier journal ainsi que d'ajouter un fichier au journal des transactions. Si la modification peut s'effectuer comme pour les fichiers de données par l'intermédiaire de la commande ALTER DATABASE nomBaseDonnées MODIFY FILE…, l'ajout quant à lui nécessite l'utilisation de l'option ADD LOG FILE. Bien entendu comme pour les fichiers de données, toutes ces opérations peuvent être effectuées depuis SQL Server Management Studio.




## b. Libérer de l'espace disque utilisé par des fichiers de données vides.

Lorsque les tables sont vidées de leurs données à l'aide des commandes DELETE ou TRUNCATE TABLE, les extensions occupées par les tables et index sont alors libérées. Par contre, la taille des fichiers n'est pas réduite. Pour réaliser une telle opération il est important de s'assurer que la totalité de l'espace libre est regroupée en fin de fichier. Une fois cette opération réalisée, il est possible de tronquer le fichier sans jamais redescendre en dessous de la taille initiale.

La mise en place de la réduction des fichiers se fait au moyen de deux commandes DBCC.

**SHRINKDATABASE**

Cette instruction permet de compacter l'ensemble des fichiers constituant la base de données (journaux et données). Pour les fichiers de données toutes les extensions utilisées sont stockées de façon contiguë en haut du fichier. Pour les fichiers journaux, cette opération de compactage intervient en différé et SQL Server essaie de rendre aux fichiers journaux une taille aussi proche possible que celle de la taille cible lorsque les journaux sont tronqués.

*Syntaxe :*

```
DBCC SHRINKDATABASE {nom_base_données|id_base_données|0}
[,pourcentage_cible]
[,{NOTRUNCATE|TRUNCATEONLY}])
```

`nom_base_données`

Nom de la base de données sur laquelle va porter l'exécution de la commande

`id_base_données`

Identifiant de la base de données. Cet identifiant peut être connu en exécutant la fonction db_id() ou bien en interrogeant la vue sys.databases depuis la base master.

0

Permet de préciser que l'exécution portera sur la base courante.

pourcentage_cible

Permet de préciser en pourcentage l'espace libre souhaité dans le fichier après compactage.

NOTRUNCATE

Permet de ne pas rendre au système d'exploitation l'espace libre obtenu après compactage. Par défaut, l'espace ainsi obtenu est libéré.

TRUNCATEONLY

Permet de libérer l'espace inutilisé dans les fichiers de données et compacte le fichier à la dernière extension allouée. Aucune réorganisation physique des données n'est envisagée : déplacement des lignes de données afin de compléter au mieux les extensions utilisées par l'objet. Dans un tel cas, le paramètre pourcentage_cible est ignoré.

**SHRINKFILE**

Cette instruction, semblable à DBCC SHRINKDATABASE permet de réaliser les opérations de compactage et réduction de fichier de données par fichier.

*Syntaxe :*

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

taille_cible

Permet de préciser la taille finale souhaitée exprimée en mégaoctets sous forme de nombre entier. Si aucune taille n'est spécifiée, la taille du fichier est réduite à son maximum.

EMPTYFILE

Cette commande permet de réaliser la migration de toutes les données contenues dans le fichier vers les autres fichiers du même groupe. De plus, SQL Server n'utilise plus ce fichier. Il est alors possible de le supprimer de la base de données à l'aide d'une commande ALTER DATABASE.

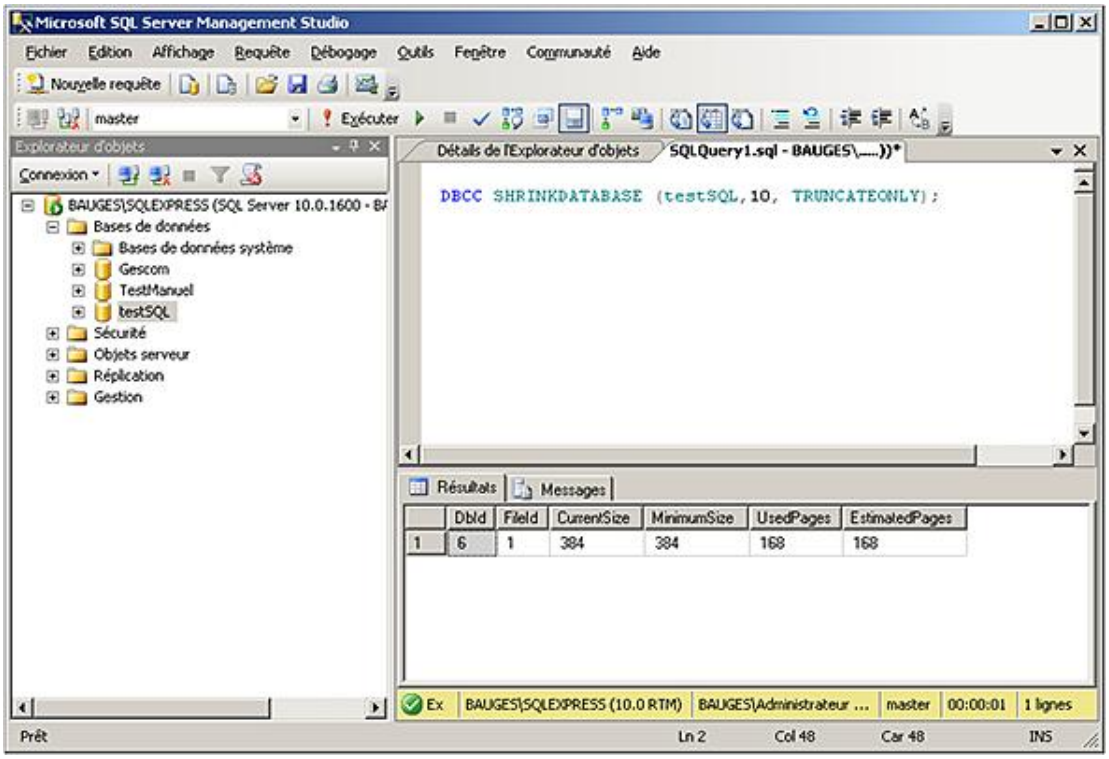
NOTRUNCATE

Permet de ne pas rendre au système d'exploitation l'espace libre obtenu après compactage. Par défaut, l'espace ainsi obtenu est libéré.

TRUNCATEONLY

Permet de libérer l'espace inutilisé dans les fichiers de données et compacte le fichier à la dernière extension allouée. Aucune réorganisation physique des données n'est envisagée : déplacement des lignes de données afin de compléter au mieux les extensions utilisées par l'objet. Dans un tel cas le paramètre taille_cible est ignoré.

*Exemple :*




La taille finale doit être supérieure à la taille de la base **Model** plus la taille des données.

---

Avant de réaliser une opération de compactage, il est prudent de réaliser une sauvegarde complète de la base de données à compacter ainsi que de la base **Master**.

---

Les instructions DBCC SHRINKDATABASE et DBCC SHRINKFILE s'exécutent en mode différé, il est donc possible que la taille des fichiers ne diminue pas de façon instantanée.

Seule l'instruction DBCC SHRINKFILE permet de réduire la taille d'un fichier à une taille inférieure à celle précisée lors de la création du fichier. Évidemment, il n'est pas possible de descendre en dessous de la taille occupée par les données.

### c. Configuration de la base de données

Il est possible de paramétrer de nombreuses options au niveau de la base de données. Ce paramétrage est possible soit avec l'instruction ALTER DATABASE en Transact SQL, soit depuis la fenêtre des propriétés de la base dans SQL Server Management Studio. Tous les paramètres listés ci-dessous sont à fixer pour chaque base de données. Il n'est donc pas possible de fixer des options sur plusieurs bases de données en une seule commande, tandis qu'il est possible de préciser plusieurs options d'une base en une commande.

Si certains choix doivent être adoptés par toutes les bases utilisateur, il est préférable de changer les options de la base **Model**, ainsi toute nouvelle base utilisateur, fonctionnera avec les paramètres définis dans **Model**.

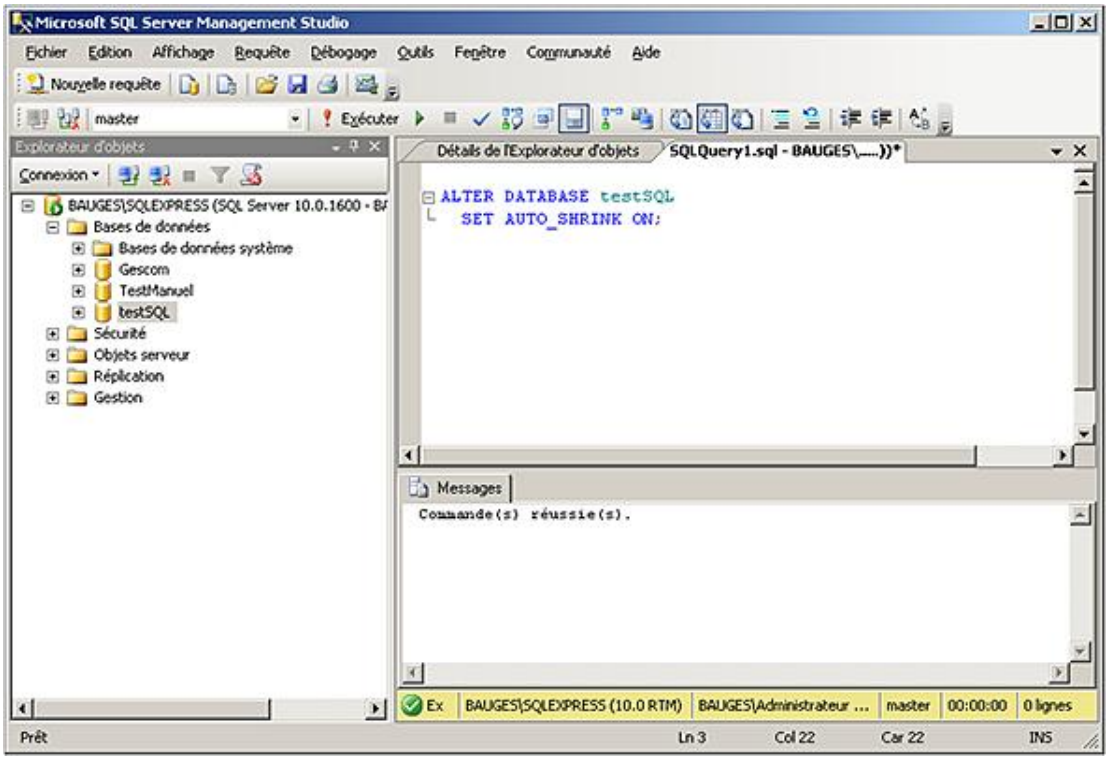
```
ALTER DATABASE nomBaseDeDonnees
SET option [;]
```

Parmi toutes les options disponibles, il est possible d'en isoler quelques-unes :

| Option | Descriptif |
|---|---|
| AUTO_SHRINK {ON\|OFF} | Si cette option est activée les fichiers sont réduits dès qu'ils disposent de plus de 25 % d'espace libre. |
| READ_ONLY | Permet de positionner la base de données en mode lecture seule. |
| READ_WRITE | La base de données est positionnée en mode lecture/écriture. |
| SINGLE_USER | Seul un utilisateur peut travailler sur la base de données. |

| | |
|---|---|
| RESTRICTED_USER | Seuls les utilisateurs membres des rôles **db_owner**, **db creator** et **sysadmin** peuvent se connecter à la base de données. |
| MULTI_USER | C'est le mode de fonctionnement standard d'une base, en autorisant plusieurs utilisateurs à travailler simultanément. |
| AUTO_CREATE_STATISTICS { ON \| OFF } | Lorsque cette option est positionnée à ON les statistiques manquantes lors de l'optimisation de la requête, sont calculées de façon automatique. |
| AUTO_UPDATE_STATISTICS { ON \| OFF} | Lorsque cette option est positionnée à ON, les statistiques obsolètes sont calculées de façon automatique. |

Les options AUTO_CREATE_STATISTICS et AUTO_UPDATE_STATISTICS ne concernent pas les tables système ou bien à usage interne à SQL Server comme par exemple les index XML, les index de texte intégral, les files d'attentes Service Broker. Pour toutes ces tables, les statistiques sont créées et mises à jour quelle que soit la valeur des options AUTO_CREATE_STATISTICS et AUTO_UPDATE_STATISTICS.

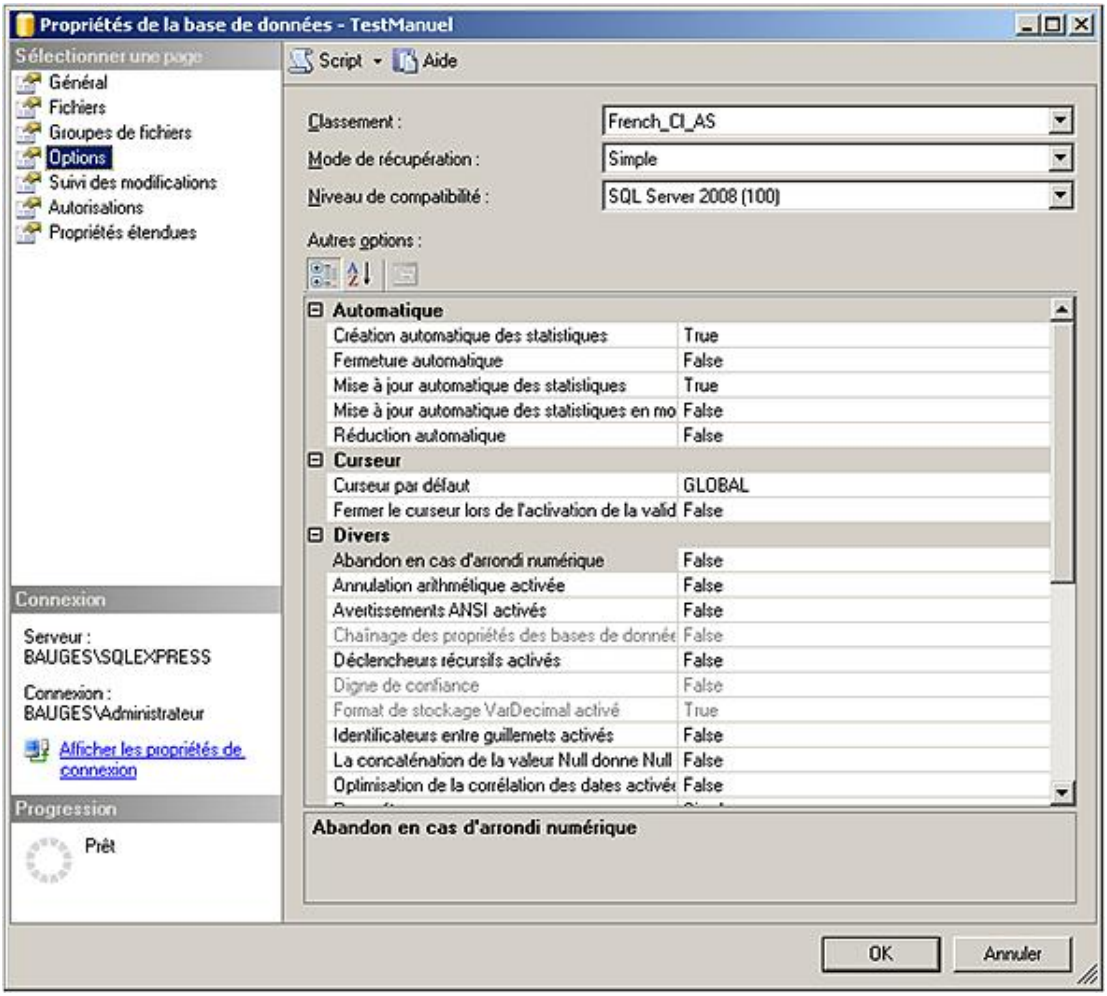



La fonction DATABASEPROPERTYEX permet de connaître la valeur actuelle de l'option qui lui est passée en paramètre.

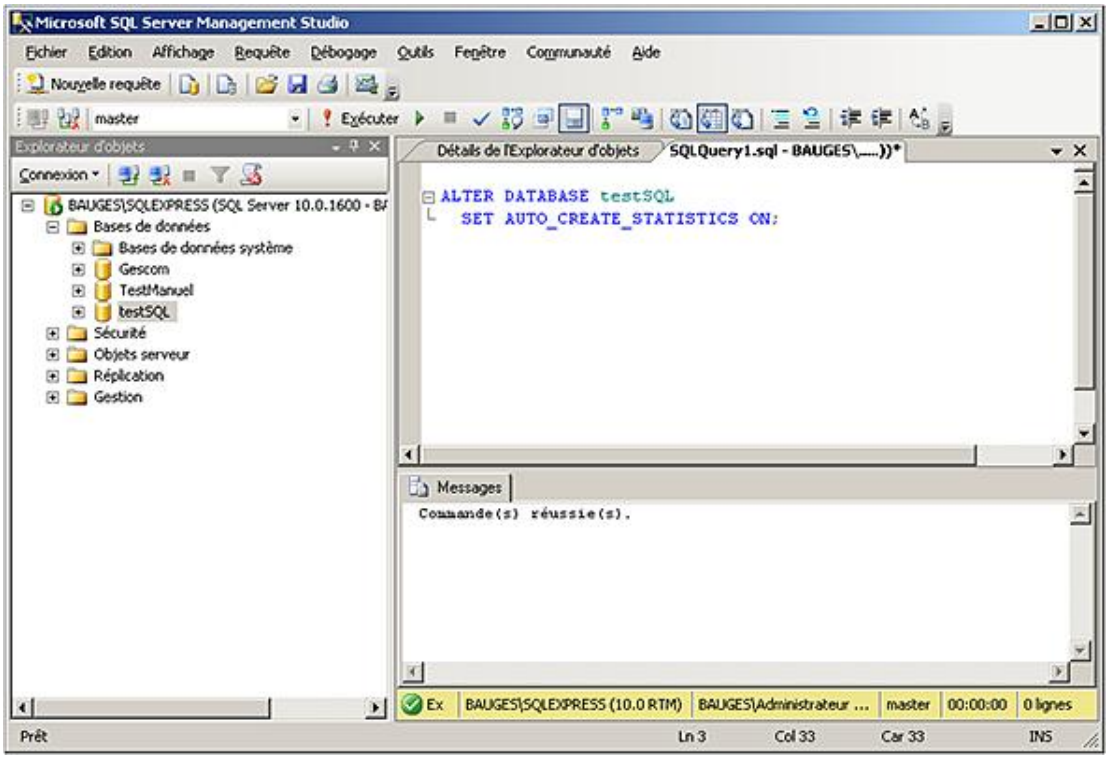
Pour régler les paramètres de la base il est possible de passer par :

## SQL Server Management Studio

Pour connaître et éventuellement modifier les paramètres d'une base de données, il faut afficher la fenêtre présentant les propriétés de la base. Cette fenêtre est affichée en sélectionnant **Propriétés** depuis le menu contextuel associé à la base de données.




## ALTER DATABASE

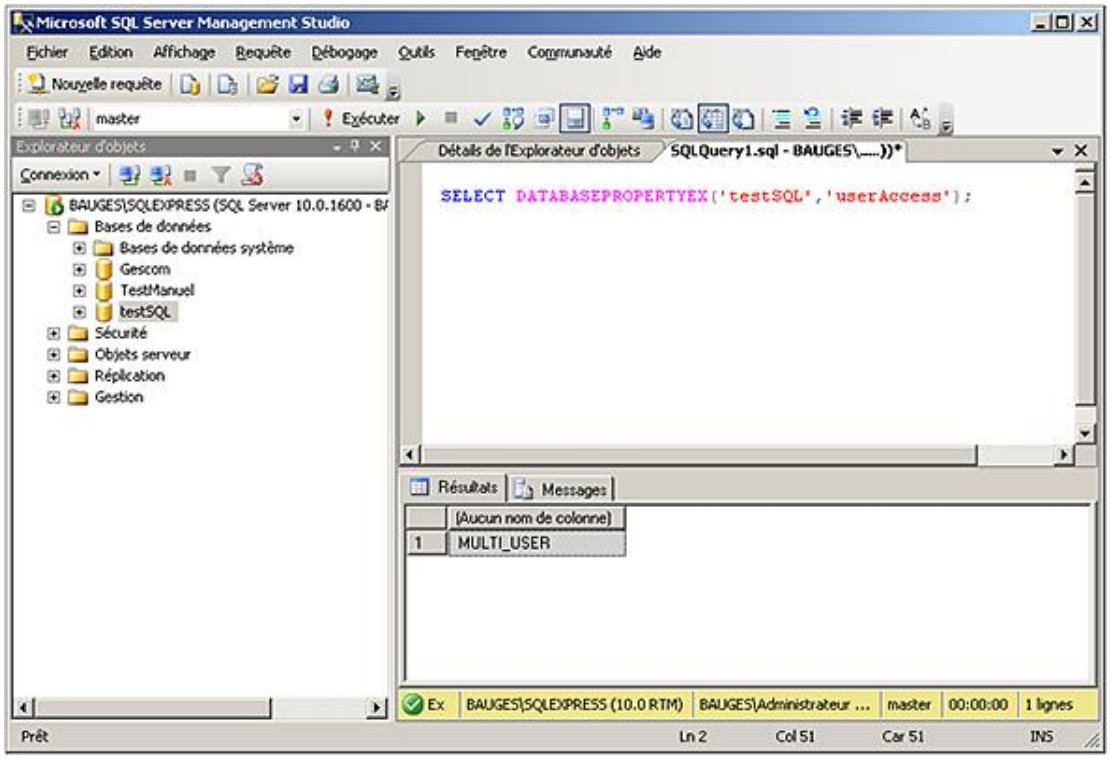



## Options actuellement gérées

Avant de fixer un nouveau paramétrage de la base de données, il est important de connaître le paramétrage actuel. La connaissance de ce paramétrage peut également aider à la compréhension du fonctionnement de la base. Il est possible de lire les valeurs des différentes options depuis SQL Server Management Studio, mais la connaissance du paramétrage de la base peut également être faite sous la forme de script Transact SQL.

**Databasepropertyex**

Pour connaître la valeur d'un paramètre.

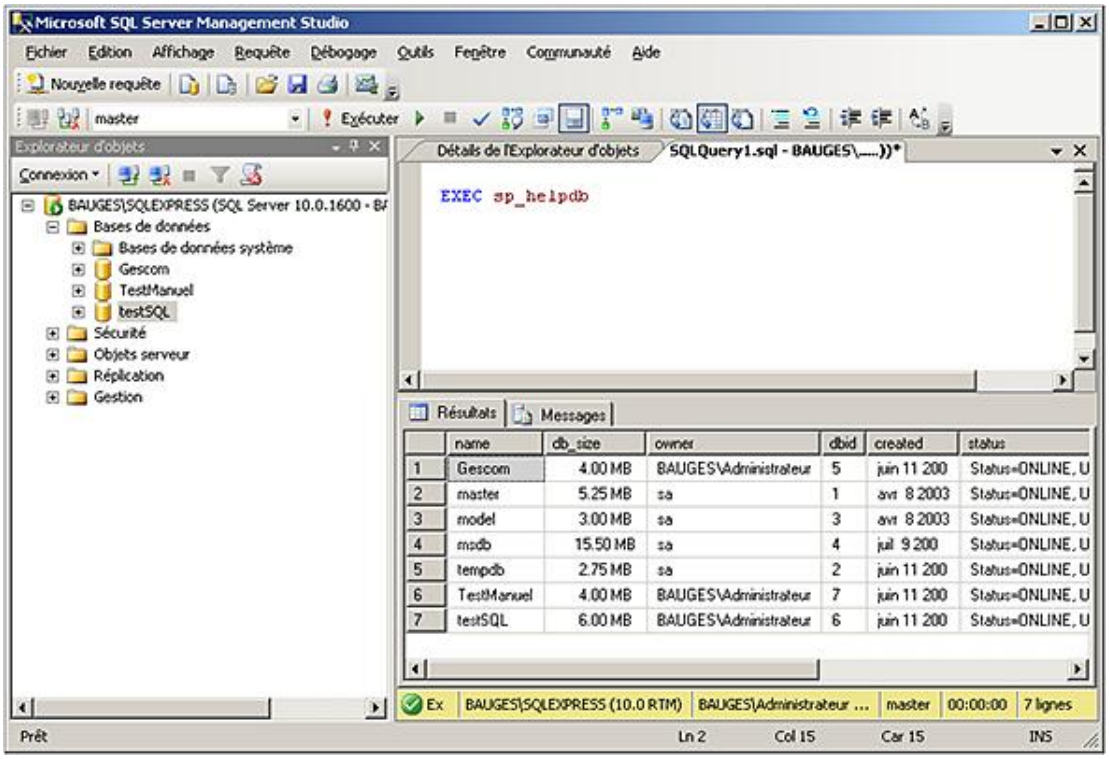



**sp_helpdb**

Si aucun paramètre n'est passé, cette procédure permet de connaître l'ensemble des bases qui existent sur le serveur. Les renseignement fournis sont le nom, la taille, le propriétaire, l'identificateur, la date de création et les options.

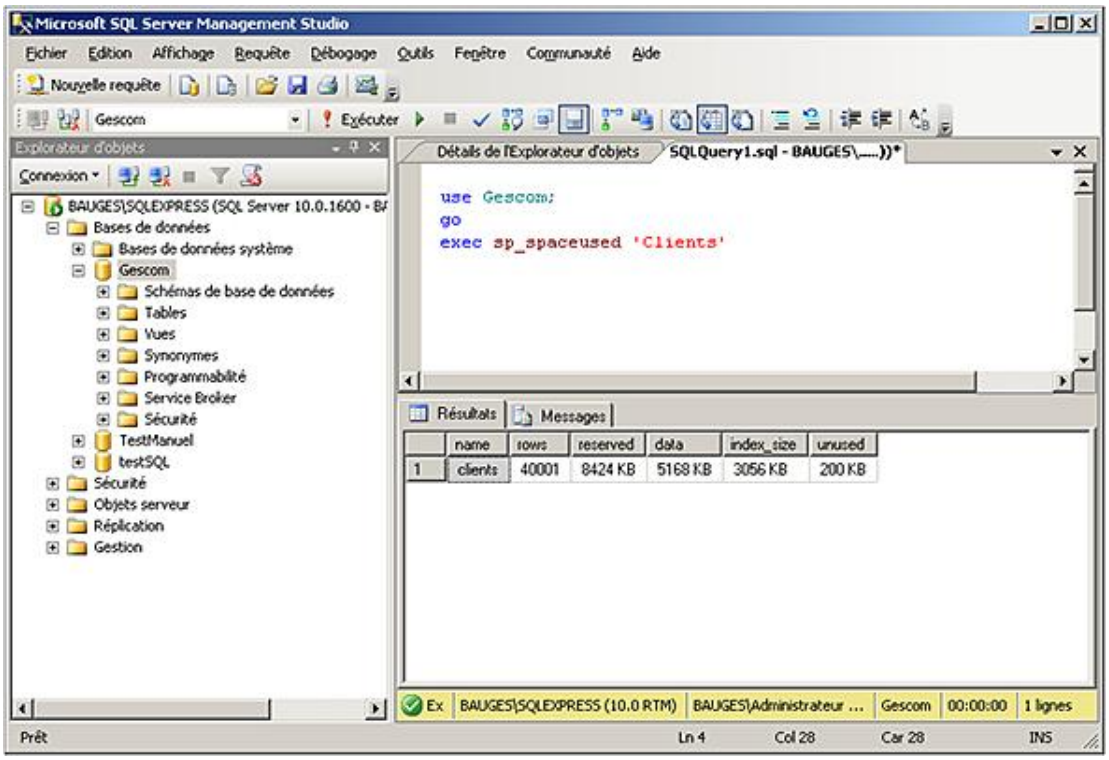
Si un nom de base est passé en paramètre, la procédure permet de connaître les informations générales de la base ainsi que les nom et emplacement des fichiers de données et des fichiers journaux.

La procédure sp_helpdb utilise la table **sys.databases** pour établir la liste des bases et les différentes options de chacune d'elles.

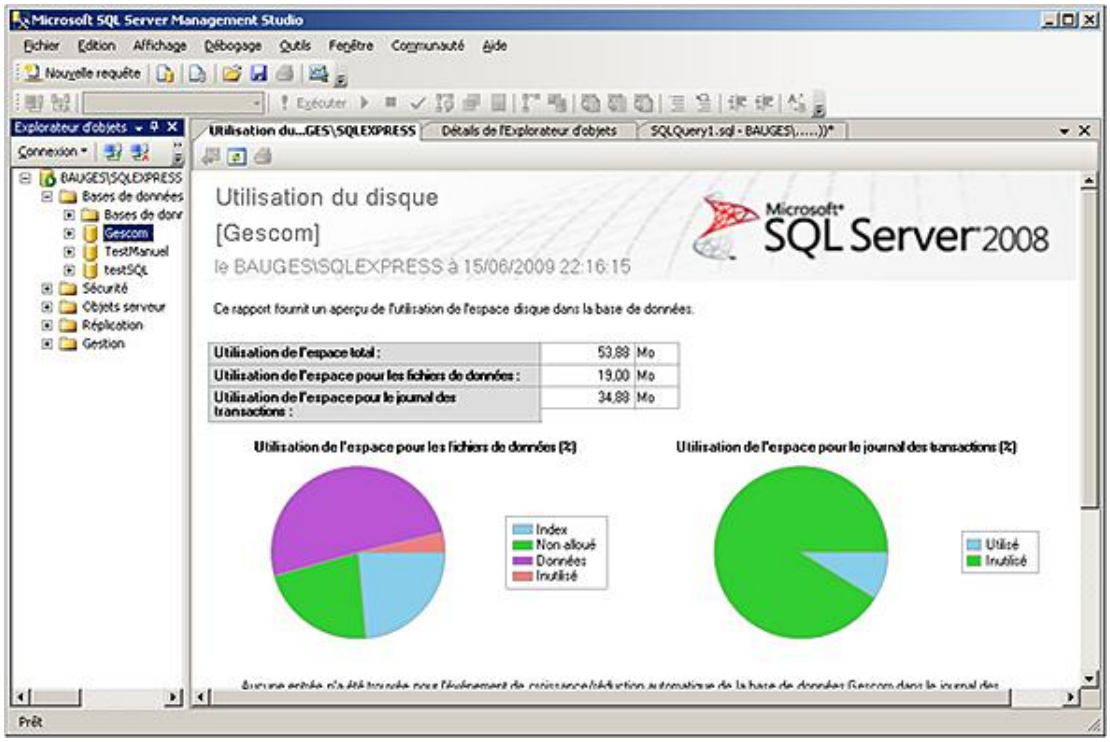



**sp_spaceused [nom_objet]**

Permet de connaître l'espace de stockage utilisé par une base de données, un journal ou des objets de la base (table...).




Il est possible au niveau de SQL Server Management Studio d'obtenir un rapport sur l'utilisation de l'espace disque par les différentes tables de la base de données en cours. Pour cela, après s'être positionné sur la base de données cible, il faut sélectionner l'option **Rapports - Rapports standards**. À ce niveau, trois choix sont disponibles en fonction que l'on souhaite un rapport global sur l'espace disque utilisé, un rapport détaillé de la consommation d'espace disque pour chaque table ou bien un rapport ne détaillant que les principales tables du serveur.




## 3. Supprimer une base de données

La suppression d'une base de données utilisateur est une opération ponctuelle qui peut être réalisée, comme la plupart des opérations d'administration soit par SQL Server Management Studio, soit en Transact SQL. La suppression de la base de données a pour conséquence de supprimer tous les fichiers qui correspondent à cette base ainsi que toutes les données contenues dans cette base. Cette opération est irréversible et en cas de mauvaise manipulation, il est nécessaire de remonter une sauvegarde.

Après suppression d'une base de données, chaque connexion qui utilisait cette base par défaut se retrouve sans base par défaut.

Afin de pouvoir réaliser d'éventuelles restaurations du serveur, il est important d'effectuer une sauvegarde de la base **Master** après suppression d'une ou plusieurs bases utilisateur.

**Les limites**

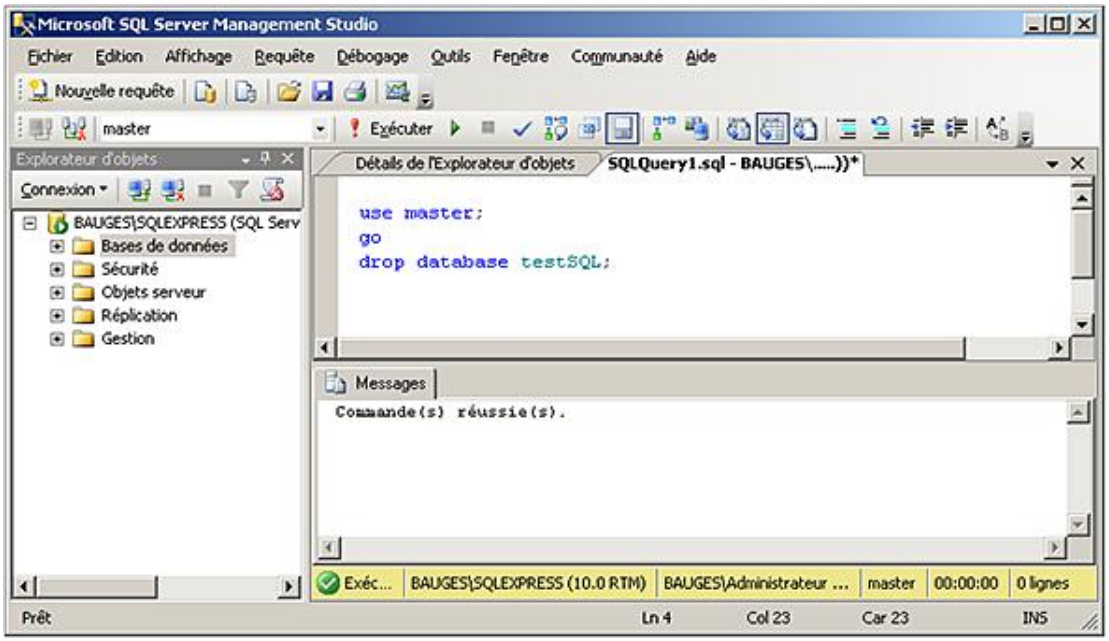
Il n'est bien sûr pas toujours possible de supprimer une base, les principales limites sont :

- lorsqu'elle est en cours de restauration,
- lorsqu'elle est ouverte par un utilisateur, en lecture ou en écriture,
- lorsqu'elle participe à une publication dans le cadre de la réplication.

Il n'est pas possible de supprimer les bases de données système.

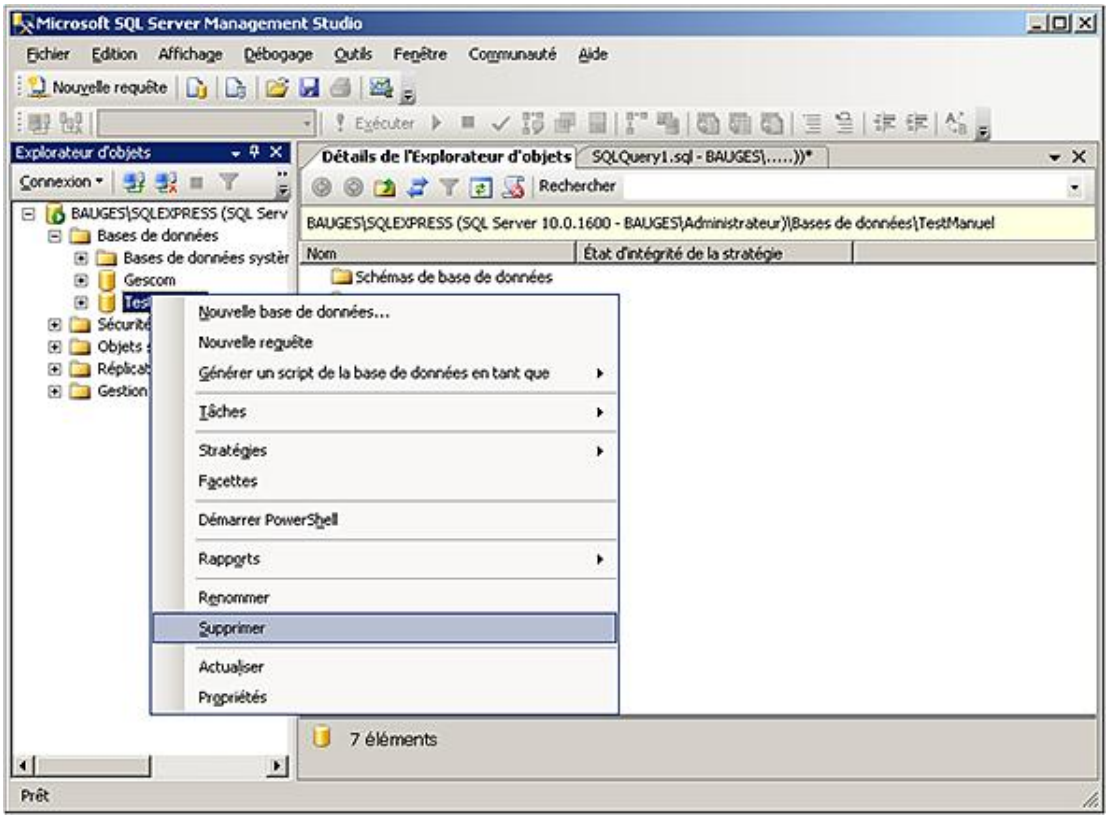
### a. Transact SQL

C'est par l'intermédiaire de la commande DROP DATABASE que seront supprimées les bases de données.




## b. SQL Server Management Studio

Par un simple clic droit sur la base de données, vous avez accès au menu **Supprimer** dans le menu contextuel.




Pour supprimer une base depuis SQL Server Management Studio, il est nécessaire de réaliser les opérations suivantes :

- Se positionner dans l'explorateur d'objets.
- Sélectionner le nœud relatif à la base de données à supprimer.
- Sélectionner l'option **Supprimer** depuis le menu contextuel associé à la base.

# Mettre en place des groupes de fichiers

Il est possible de préciser les fichiers de données utilisés par la base, mais il est malheureusement impossible de préciser sur quel fichier est créé un objet particulier. Pour résoudre ce problème, il existe la possibilité de créer une table ou un index sur un ensemble de fichiers. Cet ensemble de fichiers, également appelé **Groupe de fichiers**, est géré assez simplement.

Pourquoi est-il nécessaire de travailler avec plusieurs groupes de fichiers? Parce qu'il est normal sur un système d'exploitation de séparer les fichiers système, des programmes et des données utilisateur ; pourquoi en serait-il autrement dans une base de données ? Il convient donc de regrouper sur un même groupe de fichiers des données de même type. Par exemple, les données stables (comme par exemple les clients, les articles) des données de type mouvement (comme par exemple les commandes, les factures…) tout simplement parce que les volumes ne sont pas les mêmes, la façon de travailler avec non plus. Il peut être également intéressant de définir les index et les tables sur des groupes de fichiers distincts afin de réduire les temps de mise à jour des données et des index. Dans le cas de données sensibles, la répartition par groupe de fichiers peut également être conduite par la politique de sauvegarde adoptée.

## 1. Créer un groupe de fichiers

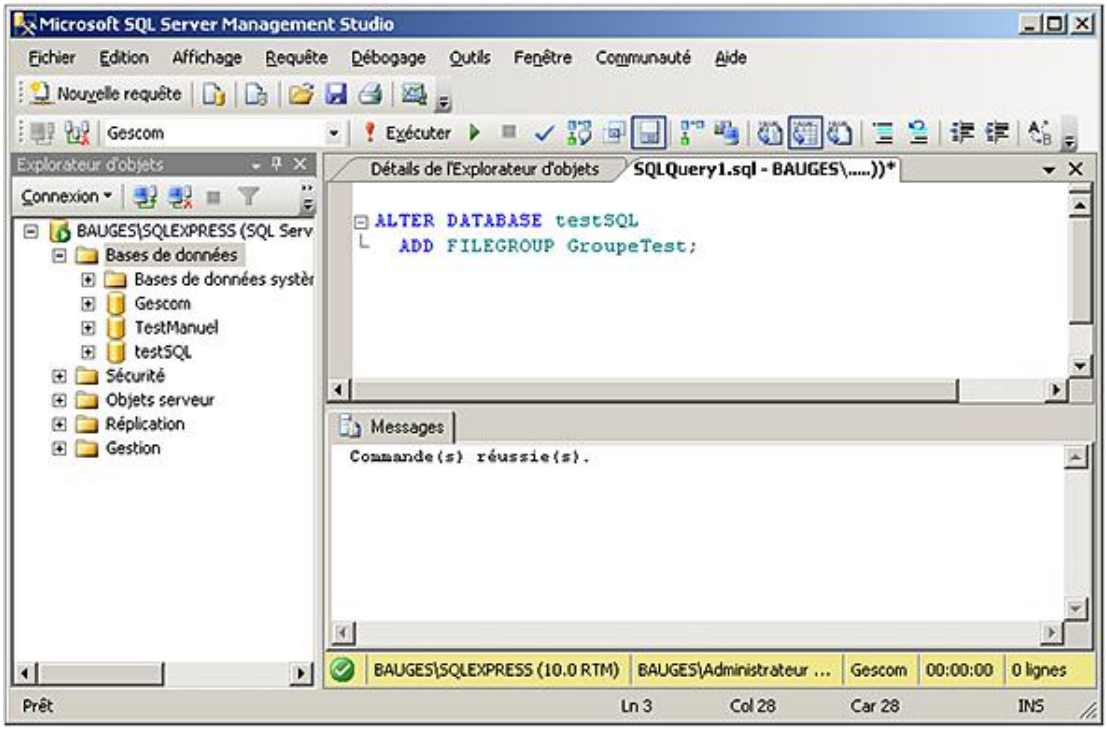
Avant de pouvoir utiliser les groupes de fichiers, il faut les créer. Lors de la création de la base, le groupe de fichiers PRIMARY a été créé.

C'est par l'intermédiaire d'une commande ALTER DATABASE que l'on va créer un groupe de fichiers. Et cette commande va permettre d'ajouter des fichiers à l'intérieur du groupe.

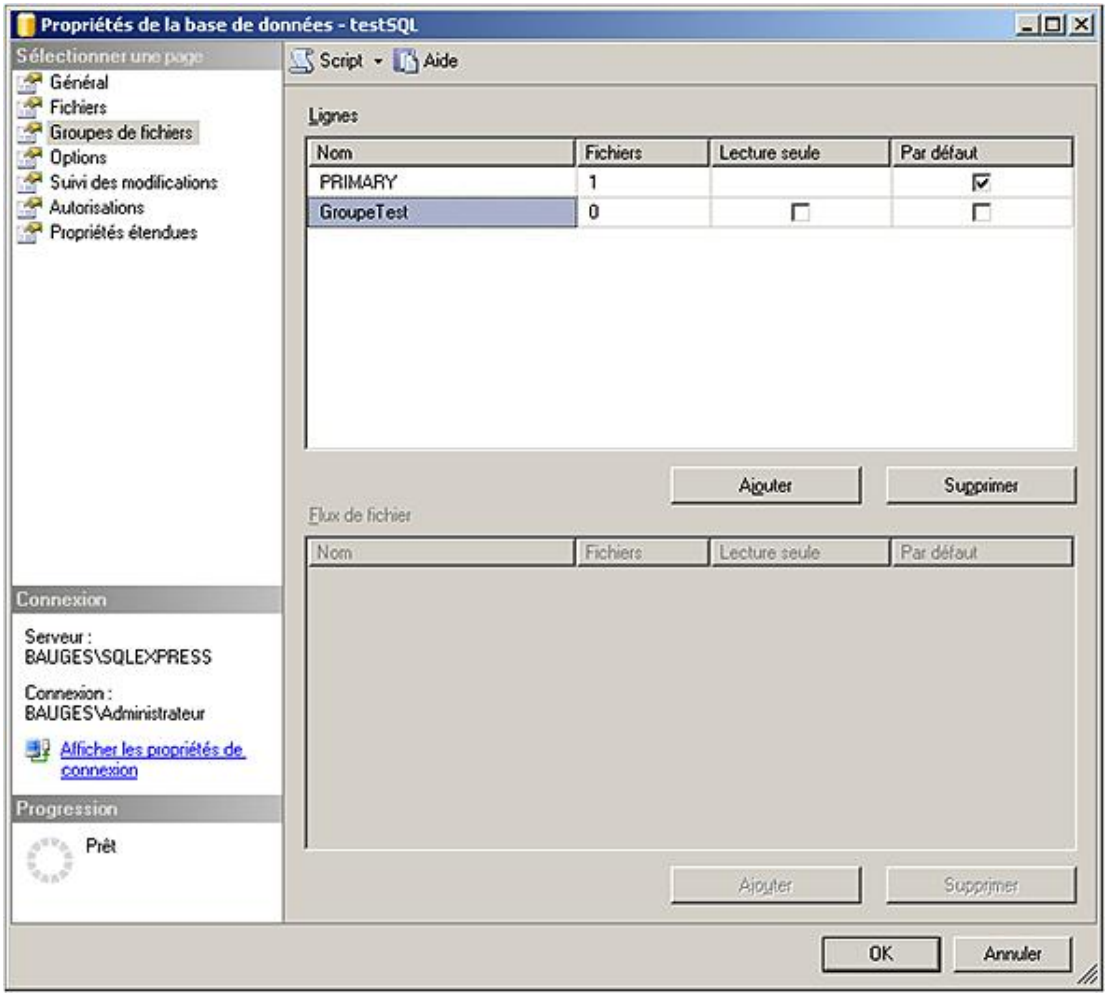
*Syntaxe :*

```
ALTER DATABASE nom_base_données
ADD FILEGROUP nom_groupe_fichiers[;]
```

*Exemple :*




Depuis SQL Server Management Studio, la gestion des groupes de fichiers est réalisée par l'intermédiaire de la fenêtre des propriétés de la base.




## 2. Ajouter des fichiers

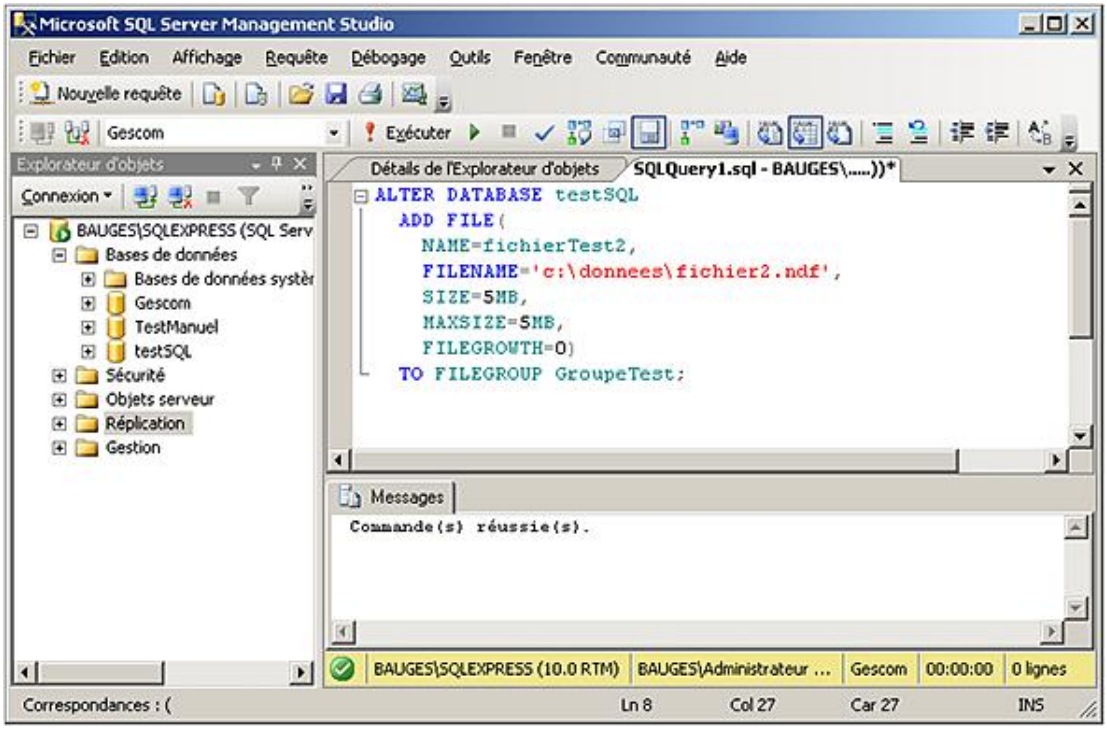
L'ajout de fichiers est réalisé par la commande ALTER DATABASE.

Un fichier ne peut appartenir qu'à un seul groupe de fichiers. Par défaut, si aucun groupe de fichiers n'est précisé lors de l'ajout du fichier à la base, il est ajouté au groupe PRIMARY.

*Syntaxe :*

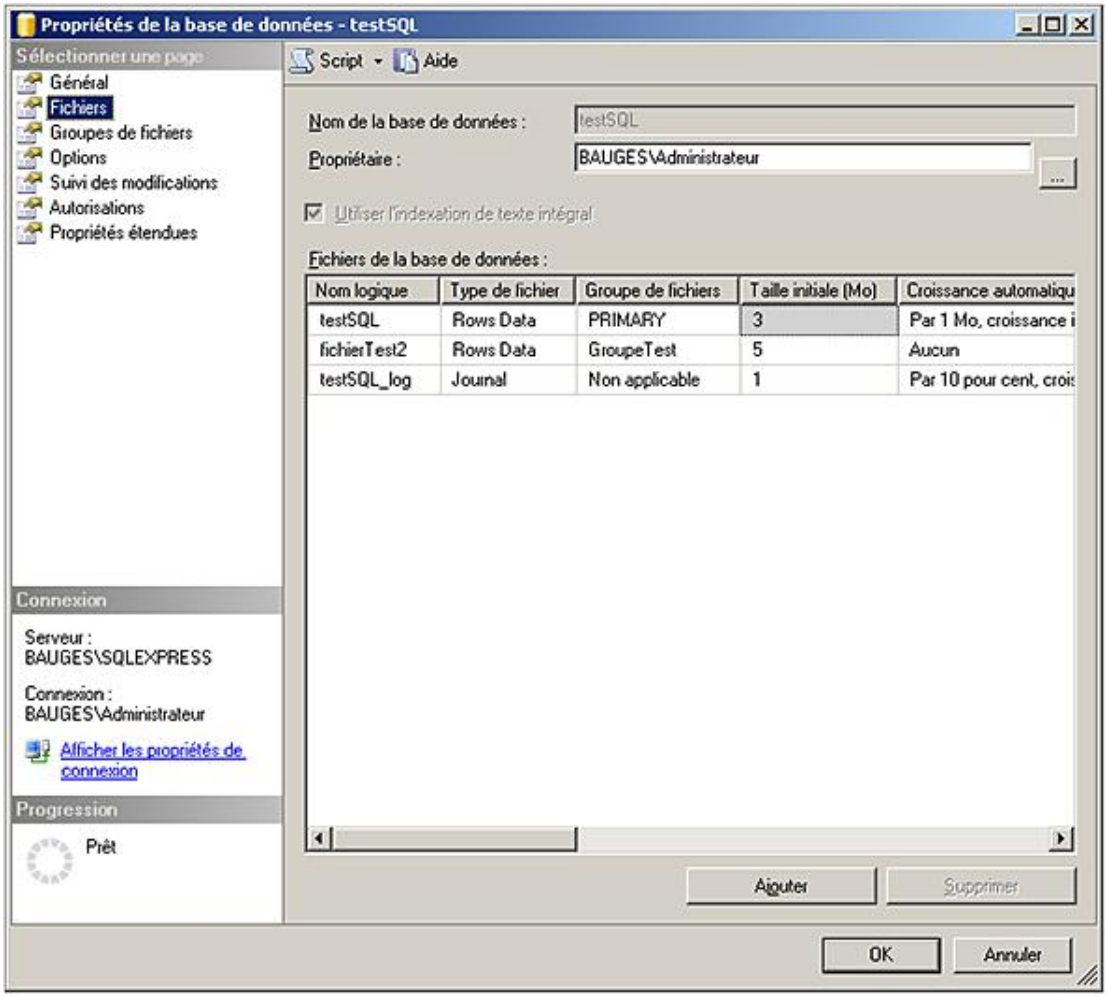
```
ALTER DATABASEnom_base_données
ADD FILE spécification fichier
TO FILEGROUP nom_groupe_fichiers
```

*Exemple :*



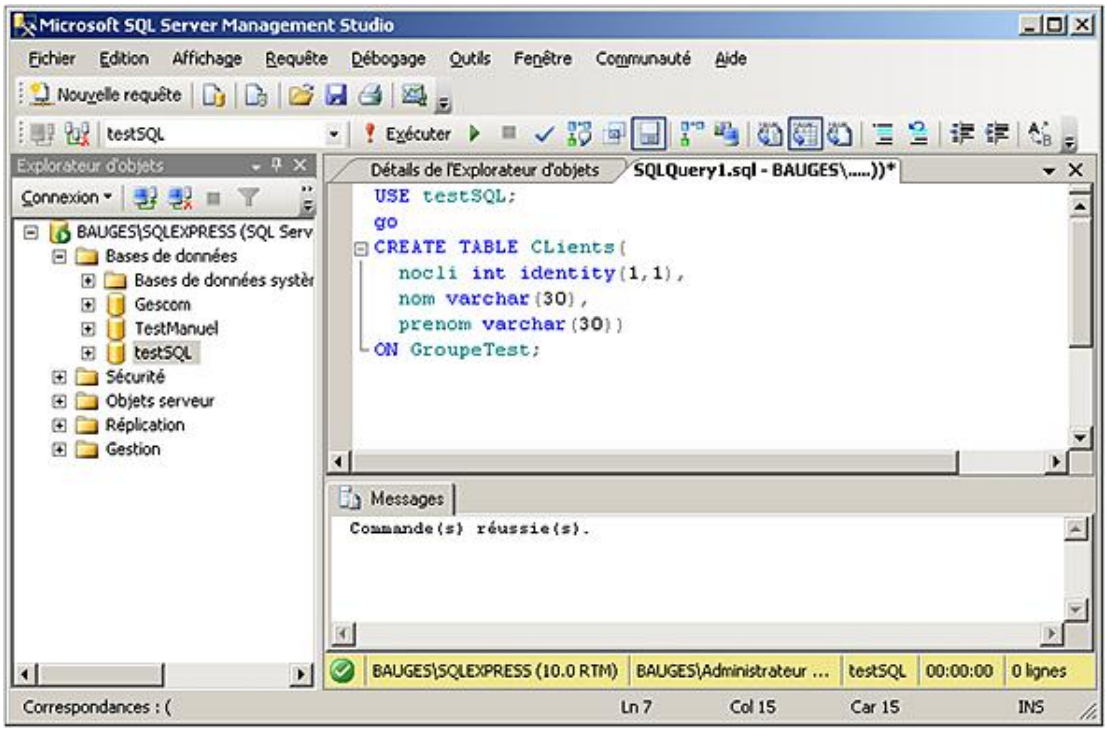

Les fichiers qui participent à un groupe de fichiers sont gérés de la même façon que ceux qui participent au groupe de fichiers **Primary**.

Depuis SQL Server Management Studio, la gestion des fichiers est possible depuis la fenêtre des propriétés de la base.



## 3. Utiliser un groupe de fichiers

L'utilisation d'un groupe de fichiers par un objet doit être précisée dès la création de ce dernier. Seuls les objets contenant de l'information sont concernés, soit les tables et les index. L'instruction `ON nom_groupe_fichiers` est simplement ajoutée à la fin de la commande SQL et avant son exécution. Un fichier ne peut appartenir qu'à un et un seul groupe de fichiers.

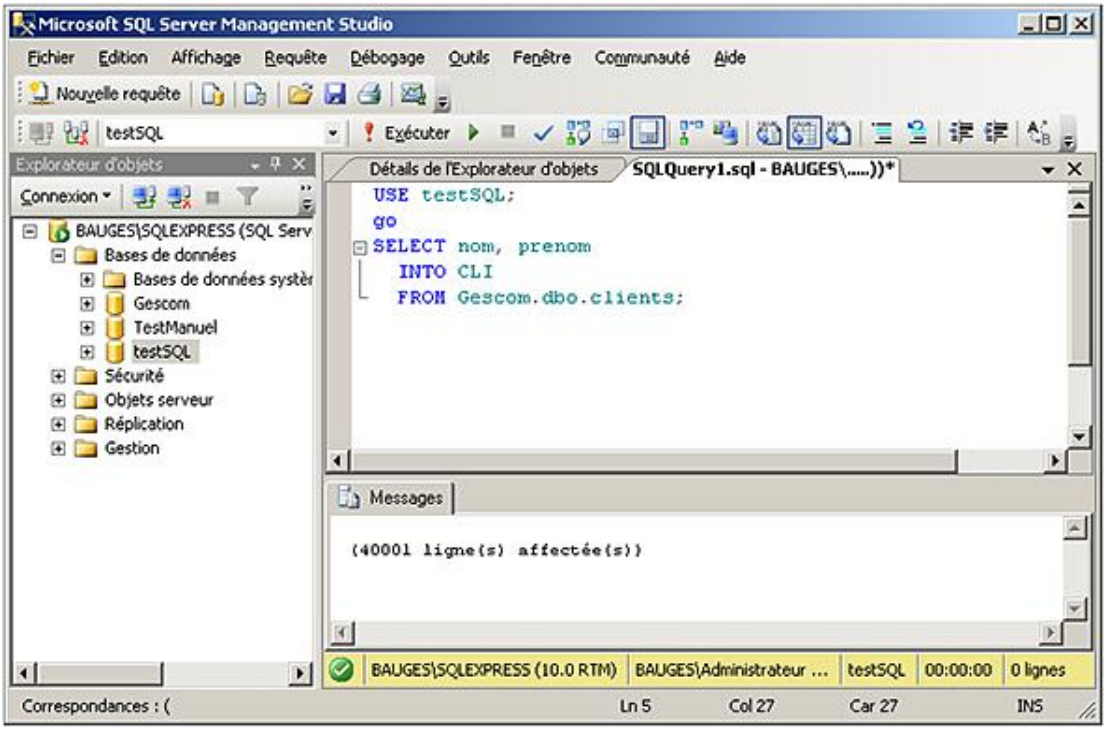



Par défaut, les objets sont créés sur le groupe PRIMARY.

# Instructions Insert, Select... into

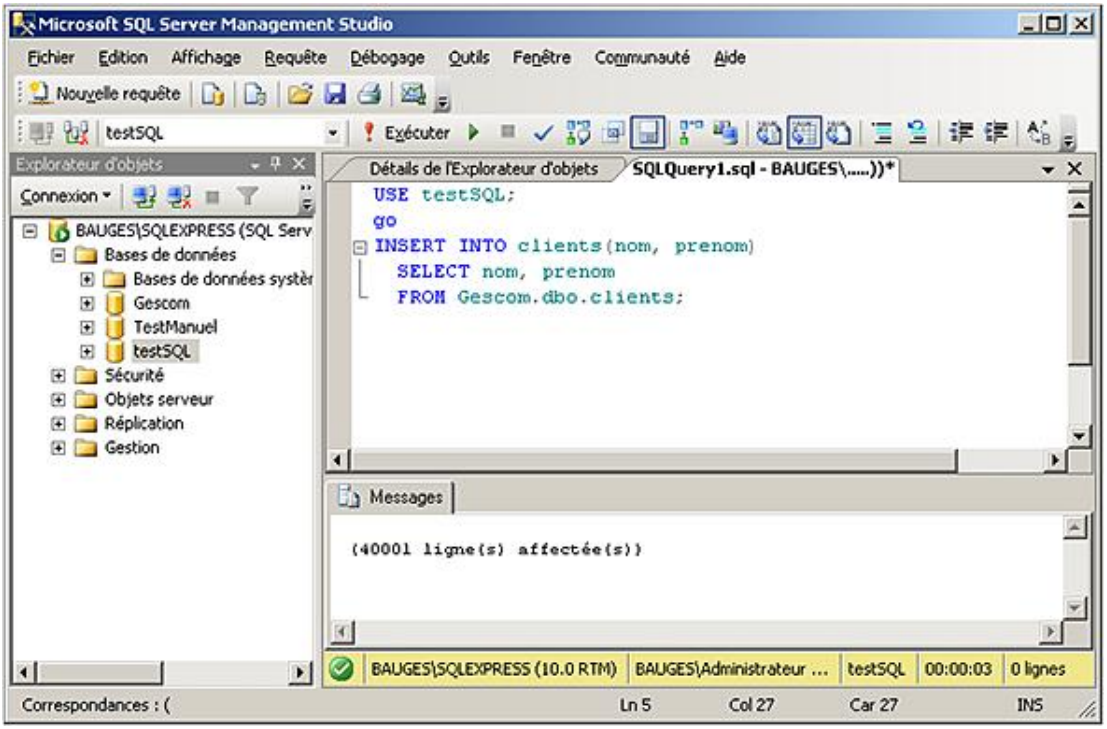
L'instruction SELECT INTO permet de projeter le résultat d'une commande SELECT dans une table. La table est créée spécialement et de façon dynamique pour contenir le résultat du SELECT.

La table créée de la sorte ne dispose d'aucune contrainte d'intégrité.




La commande INSERT est également capable de prendre en charge le résultat d'une commande SELECT afin de le projeter dans une table qui a été créée auparavant à l'aide de l'instruction SQL DDL CREATE TABLE.

Cette fois-ci, il est tout à fait possible d'inclure la définition de contraintes d'intégrité lors de la création de la table.



# Structure des index

SQL Server propose deux types d'index :

- Les index organisés ou cluster ;
- Les index non organisés ou non cluster.

Étant donné que l'index est organisé (ou ordonné) organise physiquement les données stockées dans la table, il bien souvent associé à la clé primaire de la table car il s'agit d'une données stable et peu volumineuse. Il n'est pourtant pas obligatoire. Il peut parfois être utile d'organiser selon un autre critère, par exemple lorsque la clé primaire est dénuée de sens. Chaque table possède au plus un index organisé.

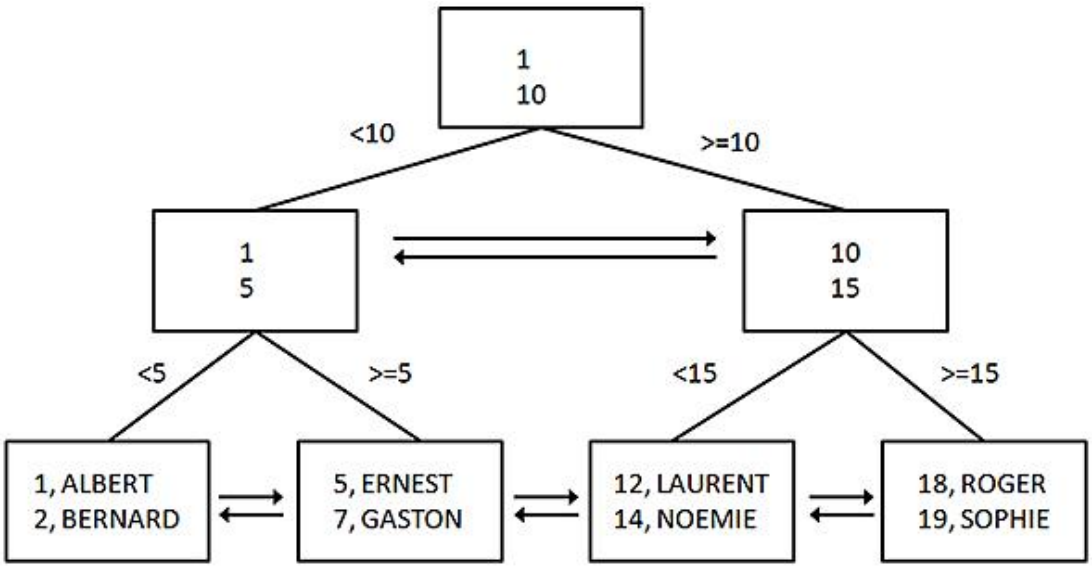
Les index non organisés, quant à eux n'affectent pas la structure physique de la table. Par contre, comme ils reposent sur l'organisation physique des données, il est nécessaire de les définir dans un second temps.

## 1. Les index ordonnés

Ces index qui organisent physiquement la table sont constitués d'un arbre dans lequel les pages de niveau feuille contiennent les données de la table sous-jacente. Les niveaux supérieurs de l'arbre permettent d'ordonner les informations par rapport à la valeur indexée. Lors de l'ajout d'une ligne d'information, cette ligne est insérée en fonction de la valeur de sa clé.

Étant donné que la clé de l'index organise physiquement la table, il est nécessaire de baser cet index sur une valeur stable et c'est pourquoi la clé primaire est traditionnellement retenue.

Le schéma ci-dessous illustre de façon synthétique la structure d'un index ordonné. En plus des chaînages permettant de parcourir l'arbre de bas en haut (depuis la racine vers les feuilles), il existe un double chaînage permettant de parcourir toutes les pages d'un même niveau.




L'index ordonné est créé par défaut lorsqu'une contrainte de clé primaire est définie sur une table. Si l'administrateur souhaite que la définition de la contrainte ne s'accompagne pas de la création d'un tel index il lui est nécessaire de spécifier le mot clé NONCLUSTERED lors de la définition de la contrainte.

*Syntaxe :*

```
ALTER TABLE nomTable
ADD CONSTRAINT nomContrainte PRIMARY KEY
[CLUSTERED|NONCLUSTERED] (listeColonnes);
```

La seconde possibilité est de définir un index avec l'option CLUSTERED

*Syntaxe :*

```
CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX nomIndex
ON nomTable (listeColonnes)
[ON groupeDeFichiers];
```

## 2. Les index non ordonnés

L'autre type d'index qu'il est possible de définir au niveau de SQL Server concerne les index dits NONCLUSTERED, c'est-à-dire que la définition de ces index ne réorganise pas physiquement la table. De ce fait, il est possible de définir plusieurs index de ce type sur une même table.

Il n'est pas possible de définir plus de 249 index non ordonnés sur une même table.

Si la création d'un index ordonné (CLUSTERED) est planifiée pour une table, il est souhaitable que cette définition intervienne avant la définition des index non ordonnés (NONCLUSTERED).

Comme pour les index ordonnés, ces index peuvent contenir une ou plusieurs colonnes, avec soit un tri ascendant (par défaut), soit descendant pour chaque colonne. L'ordre de tri est spécifié à l'aide des caractères **ASC** pour ascendant ou bien **DESC** pour descendant derrière le nom de chaque colonne.

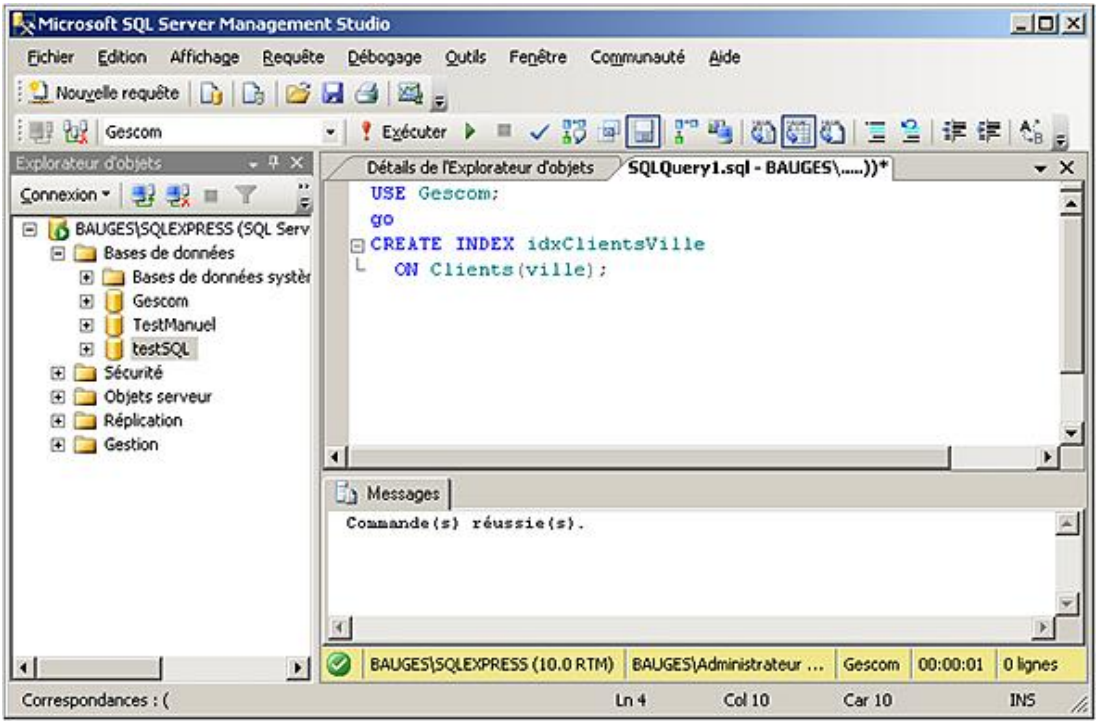
Contrairement à l'index ordonné qui doit être posé sur des données relativement stables, l'index non ordonné peut être défini sans prendre en compte la stabilité de valeurs indexées. C'est plutôt l'utilisation qui est faite des données qui va permettre de définir ces index. Les opérations de tri et de jointure peuvent être très nettement accélérées grâce à la définition de d'index.

Si un index ordonné est défini sur une table qui possède des index non ordonnés, alors les index non ordonnés sont reconstruits.

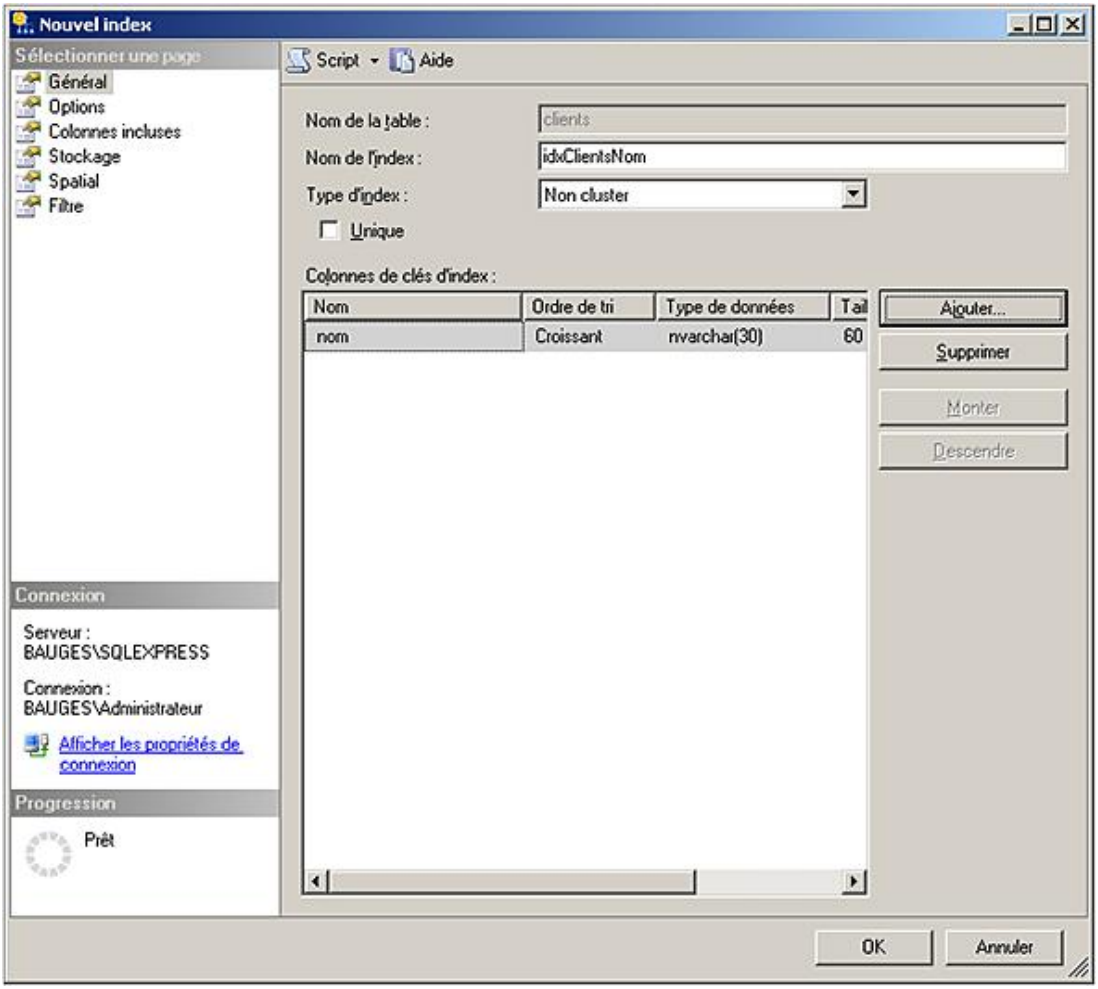
Les critères permettant de définir ou non les index peuvent être définis au travers de l'assistant paramétrage de base de données. L'utilisation de cet outil est détaillée au chapitre Optimisation.

*Exemple :*

*L'exemple suivant illustre la définition d'un index sur la colonne ville de la table des clients.*




Il est également possible de définir ce type d'index directement depuis SQL Server Management Studio.




## 3. Les index couvrants

Il s'agit cette fois d'une particularité de SQL Server, qui consiste à définir des index qui vont contenir au niveau feuille la clé de l'index ainsi que les valeurs issues d'une ou de plusieurs colonnes. L'objectif de ces index est de permettre au moteur SQL Server de ne parcourir que l'index, sans qu'il soit nécessaire d'accéder à la table pour répondre aux besoins de données d'une requête.

En terme de volume de données manipulées le gain peut être conséquent, toutefois il est à pondéré par rapport au volume disque occupé mais aussi par le temps supplémentaire nécessaire pour mener à bien les actions de mise à jour (INSERT, UPDATE et DELETE).

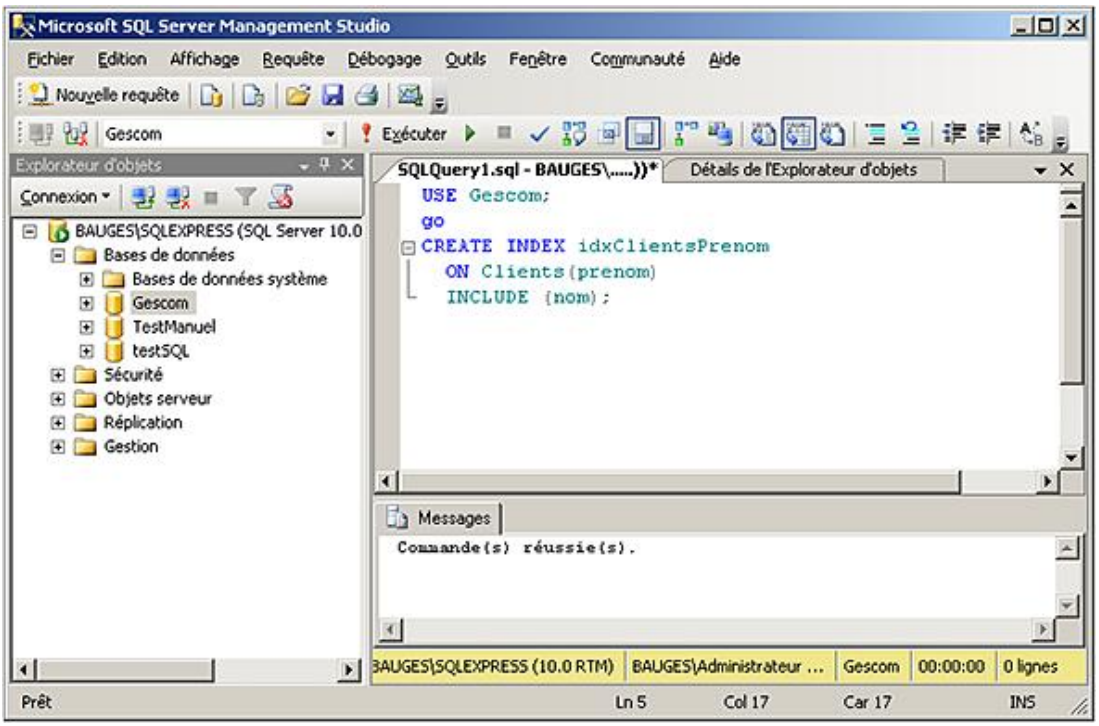
De tels index sont définis à l'aide de l'instruction CREATE INDEX suivi du mot clé INCLUDE afin de préciser la ou les colonnes à inclure au niveau colonne.

**Syntaxe :**

```
CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX nomIndex
ON nomTable (listeColonnes)
INCLUDE (listeColonnes)
[ON groupeDeFichiers];
```

*Exemple :*

*Dans l'exemple suivant, un index est défini sur le prénom du client et le nom est inclus au niveau feuille.*

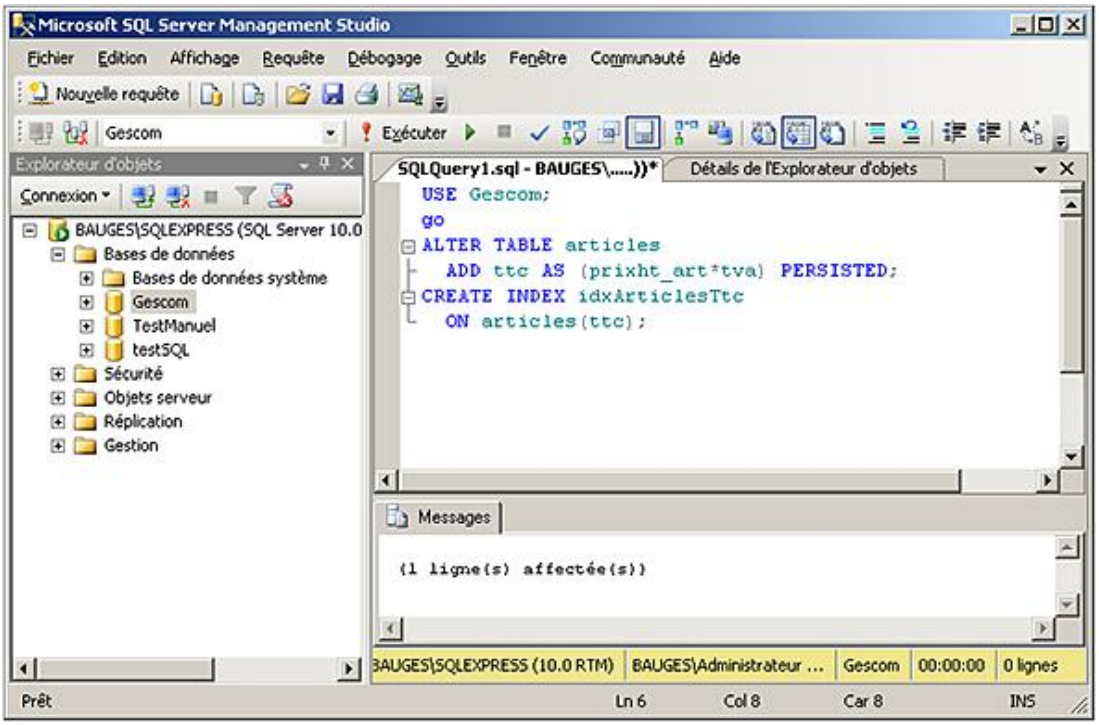



## 4. Indexer des colonnes calculées

Toujours dans l'objectif de répondre rapidement aux utilisateurs, il est possible de définir des colonnes calculées dans une table. Toutefois pour pouvoir être défini, le calcul devra être élémentaire, c'est-à-dire portant sur chaque ligne de données et non pas issu d'un regroupement. Les données doivent toutes provenir de la même table et la fonction de calcul doit être déterministe.

Dans ce cas, il est possible de définir un index sur ces colonnes calculées.

*Exemple :*



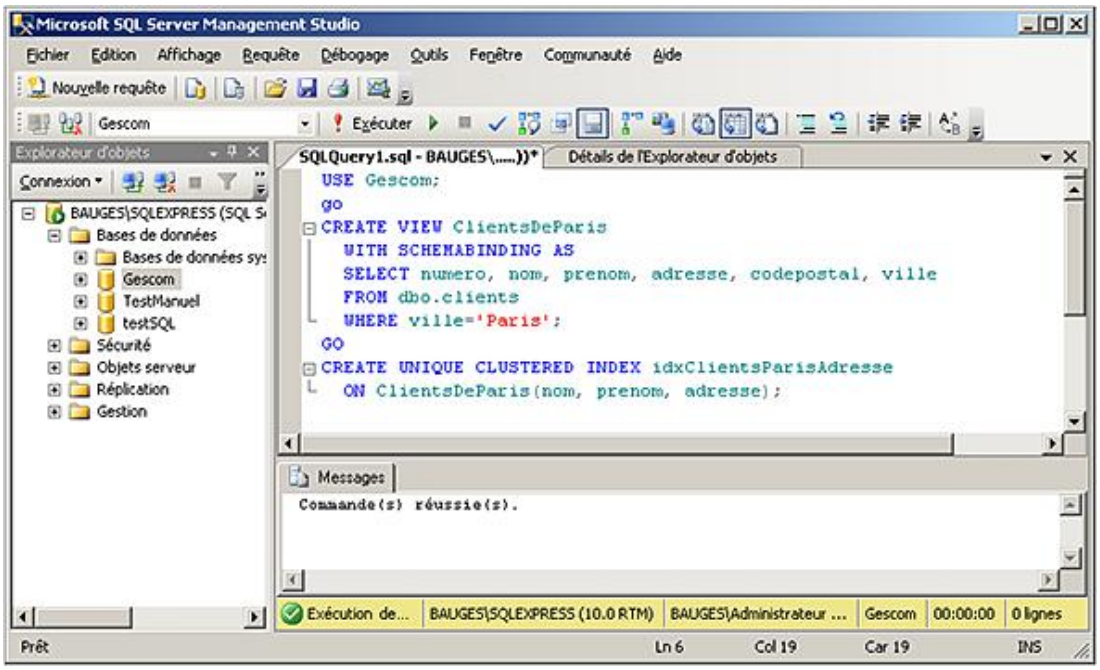
## 5. Indexer les vues

Les vues sont fréquemment utilisées dans les requêtes d'extraction car elles permettent, entre autres, de simplifier l'écriture des requêtes. Pour améliorer les performances des requêtes qui utilisent les vues, il est possible de définir un ou plusieurs index sur les vues.

Le point de départ consiste à définir un index ordonné (CLUSTERED) unique afin de matérialiser la vue. Par la suite des index non ordonnés peuvent être définis sur la vue. Même les colonnes présentant le résultat d'un calcul peuvent être indexées.

La syntaxe de définition d'un index sur une table ou bien sur une vue est exactement identique.

*Exemple :*




## 6. Les index XML

Comme pour les autres colonnes il est possible d'indexer les colonnes de type XML. Cependant, l'indexation des données XML est particulière en fonction de la structure même des données. Lors du traitement d'une requête, les informations XML sont analysées au niveau de chaque ligne, ce qui peut entraîner des traitements longs et coûteux lorsque le nombre de lignes est important et/ou quand les informations au format XML sont nombreuses.

Le mécanisme habituel d'indexation qui repose sur un arbre balancé va être utilisé pour définir l'index dit principal sur la colonne de type XML. Mais les index qui vont effectivement permettre d'accélérer le traitement des requêtes sont les index qui vont reposer sur cet index principal. Ces index secondaires sont définis par rapport aux types de requêtes fréquemment exécutées:

- index PATH pour des requêtes portant sur le chemin d'accès ;
- index PROPERTY pour des requêtes portant sur les propriétés ;
- index VALUE pour des requêtes portant sur des valeurs.

Il est également possible de définir un index de type texte intégral sur les colonnes de type XML.

### a. Index principal

L'index principal représente le point de départ à toute indexation de la colonne de type XML. Il ne peut être défini que sur une table qui possède une contrainte de clé primaire associée à un index qui organise physiquement la

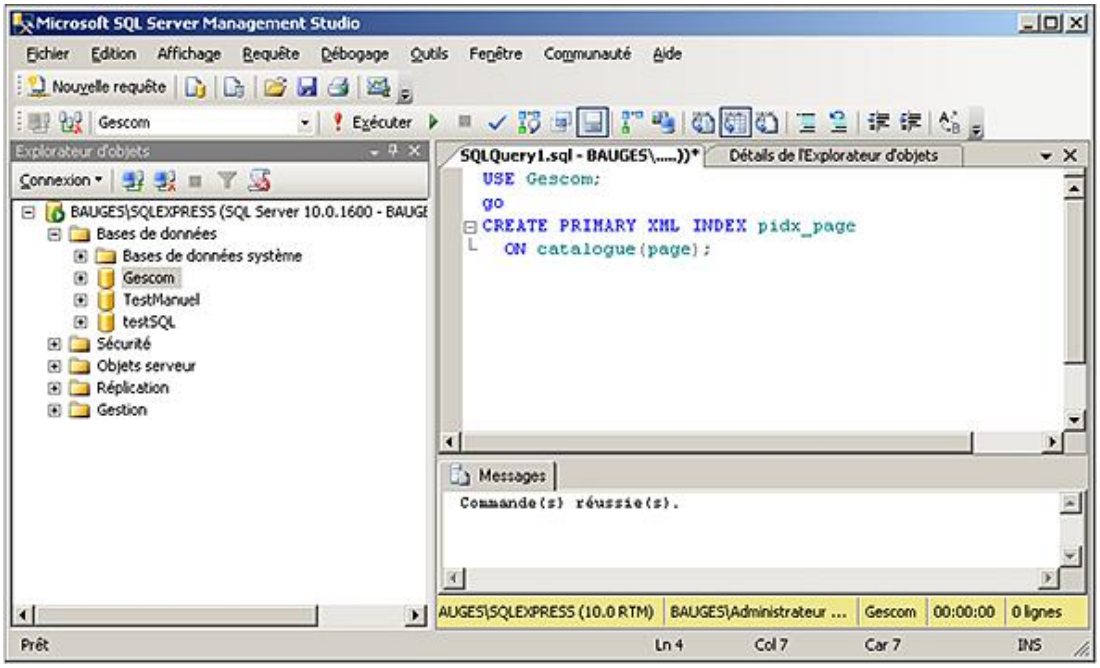
table. Ce qui est fréquemment le cas.

*Syntaxe :*

```
CREATE PRIMARY XML INDEX nomIndex ON table(colonneXML)[;]
```

*Exemple :*

*Un index principal est défini sur la colonne page de la table catalogue.*




## b. Index secondaire

La définition d'un index secondaire n'est possible que si et seulement si un index primaire est déjà défini sur la colonne.

Le document XML ne peut contenir que 128 niveaux au maximum. Les documents qui possèdent une hiérarchie plus complexe sont rejetés lors de l'insertion ou de la modification des colonnes.

L'indexation, quant à elle, porte sur les 128 premiers octets du nœud. Les valeurs plus longues ne sont pas prises en compte dans l'index.

**Syntaxe :**

```
CREATE XML INDEX nomIndex ON table(colonneXML)
USING XML INDEX nomIndexXMLPrincipal
FOR {PATH|PROPERTY|VALUE}[;]
```

PATH

Permet de construire un index sur les colonnes path et value (chemin et valeur) de l'index XML principal. Un tel index peut participer à améliorer sensiblement les temps de réponses lors de l'utilisation de la méthode **exist()**, dans une clause where par exemple.

PROPERTY

Permet de construire un index sur les colonnes PK, path et value de l'index XML principal. Le symbole PK correspond à la clé primaire de la table. Ce type d'index est donc utile lors de l'utilisation de la méthode **value()** dans les requêtes SQL de manipulation de données.

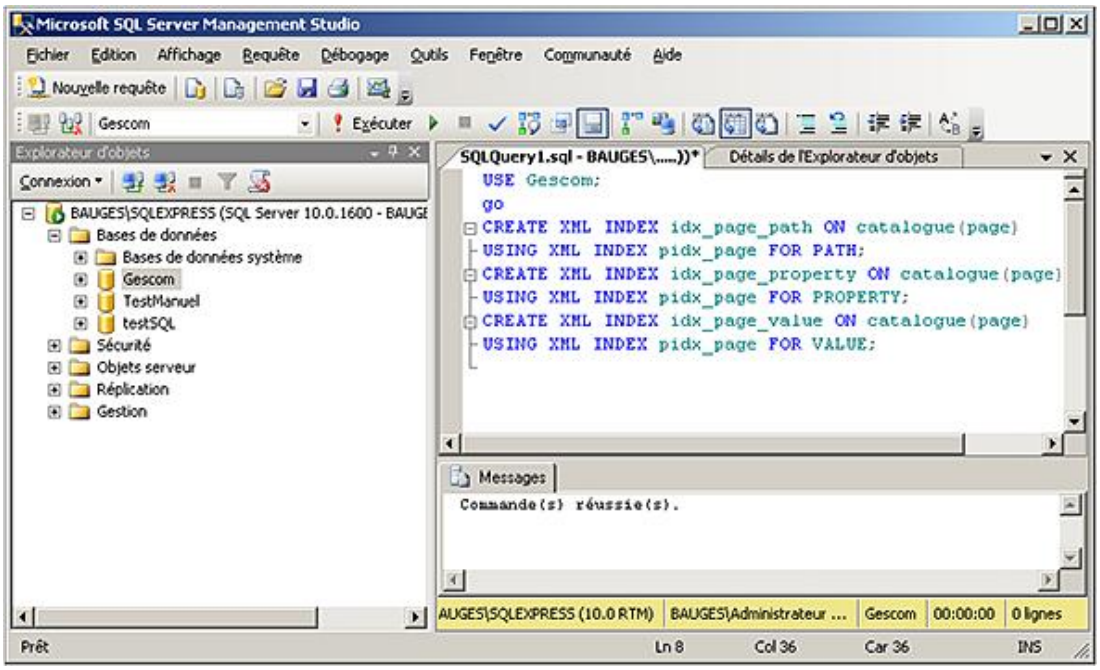
VALUE

Permet de construire un index sur les colonnes value et path de l'index XML principal. Ce type d'index est utilisé dans

les requêtes pour lesquelles la valeur du nœud est connue indépendamment du chemin d'accès, ce qui peut être le cas lors de l'utilisation de la méthode **exist()** par exemple.

*Exemple :*

*Dans l'exemple présenté ci-après, les trois types d'index sont créés par rapport à l'index XML principal défini sur la colonne page de type XML de la table Catalogue.*




## 7. Les index spatiaux

SQL Server 2008 permet de stocker des données spatiales de type geometry ou geography. Comme pour toutes les informations conservées pas SQL Server, le moteur se doit de fournir un accès rapide aux informations. Dans cet objectif les index jouent un rôle crucial. Il est donc tout à fait logique que SQL Server propose d'indexer les colonnes de type spatial.

La structure de ces index diffère des index classiques. Pour ce type d'index, SQL Server définit une structure compatible avec les données spatiales en définissant un maillage sur quatre niveaux afin d'accéder rapidement à la cellule souhaitée. Chaque niveau correspond à un maillage défini sur 16, 64 ou 256 cellules. Chaque cellule d'un niveau est détaillée par une grille de niveau inférieur. Par exemple, si le choix est fait de travailler avec une grille de 16 cellules au niveau 1 alors le niveau 4 (le plus détaillé) comportera 65536 cellules. Le nombre de cellules définies dans les grilles des différents niveaux représente la densité de l'index. Cette densité peut prendre les valeurs LOW (16 cellules), MEDIUM (64 cellules) ou bien HIGH (256 cellules).

Les zones géographiques contenues dans la table sont ainsi indexées et le parcours des différents niveaux de l'index permet de localiser rapidement la ou les cellules qui contiennent la zone cible de la recherche.

Afin de limiter l'étendue des grilles d'index, lors de la définition de l'index, la zone géographique à prendre en compte pour l'indexation est définie à l'aide de l'option BOUNDING BOX.

*Syntaxe :*

```
CREATE SPATIAL INDEX nomIndex
ON nomTable(nomColonne)
WITH (
BOUNDING_BOX=(xMin, yMin, xMax, yMAX),
GRIDS(LEVEL_1=densité1, LEVEL_2=densité2, LEVEL_3=densité3, LEVEL_4=densité4)
);
```

Ou

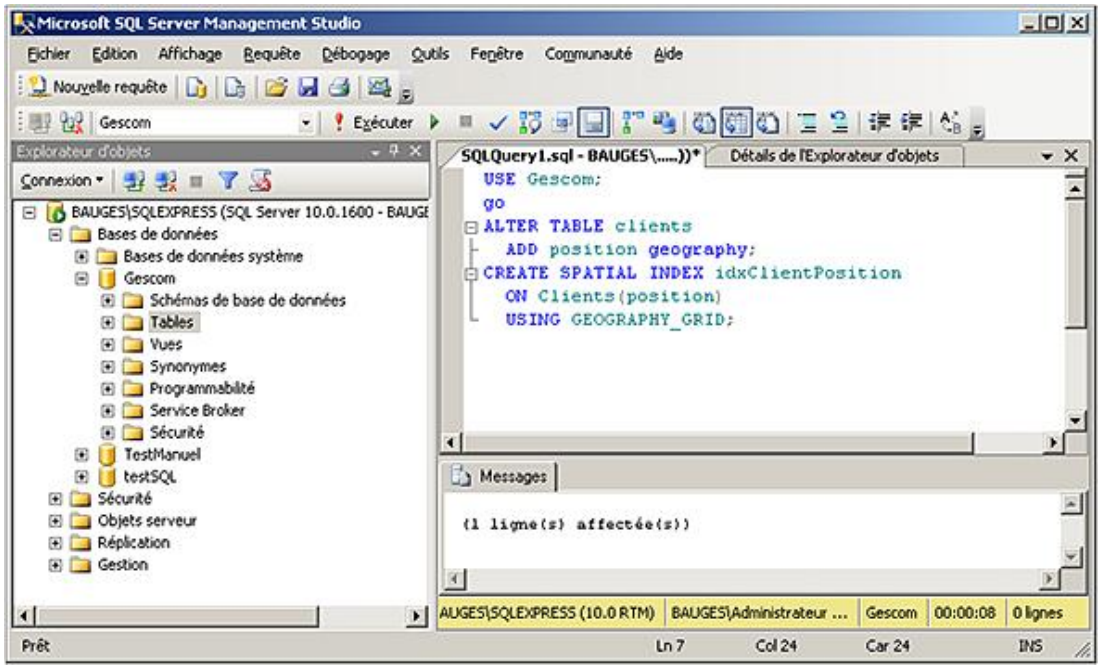
```
CREATE SPATIAL INDEX nomIndex
ON nomTable(nomColonne)
USING {GEOMETRY_GRID|GEOGRAPHY_GRID}
```

```
WITH (
BOUNDING_BOX=(xMin, yMin, xMax, yMAX),
GRIDS(densité)
);
```

L'option USING permet de spécifier lors de la création de l'index si les données indexées sont de type géométrique ou bien géographique.

Comme précisé précédemment, la densité peut prendre les valeurs LOW, MEDIUM ou HIGH. Si le niveau de densité n'est pas spécifié lors de la création de l'index alors un index est défini avec une densité de type MEDIUM par défaut.

*Exemple :*




Les index de types spatiaux peuvent être définis sur un groupe de fichiers différents de celui qui contient la table indexée.

# Planification

## 1. Dimensionner les fichiers

Afin d'évaluer la taille des fichiers nécessaires au stockage des informations contenues dans la base il faut prendre en compte de nombreux critères.

**Pour les fichiers de données**

- distinguer les tables système et utilisateur,
- prendre en compte le nombre de lignes dans les tables,
- recenser les valeur indexées (clé, nombre de ligne, facteur de remplissage).

C'est après une fine évaluation de la quantité d'espace occupée qu'il est possible de fixer la taille initiale des fichiers de données. La méthode la plus simple est d'évaluer la longueur moyenne d'une ligne, de calculer combien de lignes peuvent être stockées dans un bloc de 8 Ko, et enfin de trouver le nombre de blocs nécessaires pour stocker toutes les lignes de la table. À partir de ce nombre de blocs utilisés par la table, il convient de prendre le multiple de 8 immédiatement supérieur puis de diviser par 8 pour obtenir le nombre d'extensions.

**Pour les fichiers journaux**

- l'activité,
- la fréquence,
- la taille des transactions,
- les sauvegardes.

C'est la prise en compte de ces critères, ainsi que la consultation des options de la base, qui vont permettre de fixer une taille optimale pour le fichier journal. Au départ il peut être utile de fixer la taille du journal entre 10 % et 25 % de la taille des données dans la base. Ce pourcentage est à diminuer si la base supporte principalement des requêtes de sélection SELECT, qui n'utilisent pas le journal.

## 2. Nommer la base et les fichiers de façon explicite

Il est important de prévoir le nom de la base, les noms logiques, physiques et la taille des fichiers ainsi que la gestion dynamique ou non de l'accroissement.

## 3. Emplacement des fichiers

L'emplacement des fichiers données et journaux doit être spécifié avec précision afin de réduire au maximum les conflits d'accès au disque, et de faire travailler tous les disques de la machine de façon équitable.

## 4. Utilisation des groupes de fichiers

Il peut être intéressant, afin de limiter les conflits d'accès disque, de créer plusieurs groupes de fichiers sur le serveur et de spécialiser chacun d'eux. Un bon exemple consiste à laisser sur le groupe de fichiers PRIMARY, l'ensemble des tables système de la base et de créer un deuxième groupe pour les tables utilisateur. Une telle méthode permet de séparer les données et les tables système. Il est parfois envisageable de créer un troisième groupe pour les index.

# Introduction

Le contrôle d'accès représente une opération importante au niveau de la gestion de la sécurité sur un serveur de bases de données. La sécurisation des données nécessite une organisation des objets de façon indépendante des utilisateurs, ce qui est possible par les schémas. La sécurité passe également par un meilleur contrôle des autorisations et la possibilité d'accorder juste les privilèges nécessaires à chaque utilisateur pour qu'il puisse travailler de façon autonome.

Pour l'organisation de cette politique de sécurité, il faut prendre en compte l'organisation hiérarchique des éléments de sécurité, de façon à rendre la gestion des droits d'accès simple et efficace.

SQL Server s'appuie sur trois éléments clés qui sont :

- les entités de sécurité ;
- les sécurisables ;
- les autorisations.

Les entités de sécurité sont des comptes de sécurité qui disposent d'un accès au serveur SQL.

Les sécurisables représentent les objets gérés par le serveur. Ici, un objet peut être une table, un schéma ou une base de données par exemple.

Les autorisations sont accordées aux entités de sécurité afin qu'elles puissent travailler avec les sécurisables.

L'organisation hiérarchique permet d'accorder une autorisation (par exemple SELECT) sur un sécurisable de niveau élevé (par exemple le schéma) pour permettre à l'entité de sécurité qui reçoit l'autorisation d'exécuter l'instruction SELECT sur toutes les tables contenues dans le schéma.

Les vues du catalogue système permettent d'obtenir un rapport complet et détaillé sur les connexions existantes, les utilisateurs de base de données définis et les privilèges accordés. Quelques-unes de ces vues sont présentées ci-dessous :

- sys.server_permissions : liste des permissions de niveau serveur et de leurs bénéficiaires.
- sys.sql_logins : liste des connexions
- sys.server_principals : entité de sécurité définie au niveau serveur
- sys.server_role_members : liste des bénéficiaires d'un rôle de serveur.
- sys.database_permissions : liste des permissions et de leur bénéficiaire au niveau base de données.
- sys.database_princpals : entité de sécurité de niveau base de données.
- sys.database_role_members : liste des bénéficiaires d'un rôle de base de données.

Pour simplifier la gestion des droits d'accès, il est possible d'utiliser trois types de rôles. Les **rôles de serveur** qui regroupent des autorisations au niveau du serveur, ces autorisations sont valables pour toutes les bases installées. Les **rôles de base de données**, regroupent quant à eux des droits au niveau de la base de données sur laquelle ils sont définis. Et enfin les **rôles d'applications**, définis sur les bases de données utilisateur, permettent de regrouper les droits nécessaires à la bonne exécution d'une application cliente.

# Gestion des accès Serveur

Avant de pouvoir travailler avec les données gérées par les bases, il faut dans un premier temps se connecter au serveur SQL. Cette étape permet de se faire identifier par le serveur SQL et d'utiliser par la suite tous les droits qui sont accordés à notre connexion. Il existe dans SQL Server deux modes de gestion des accès au serveur de base de données.

**Attention**, dans cette section, seule la partie connexion au serveur est abordée. Il est important de bien distinguer la connexion au serveur et l'utilisation de bases de données. La connexion au serveur permet de se faire identifier par le serveur SQL comme un utilisateur valide, pour utiliser, par la suite, une base de données : les données et les objets. L'ensemble de ces droits seront définis ultérieurement. Ces droits sont associés à un utilisateur de base de données, pour lequel correspond une connexion.

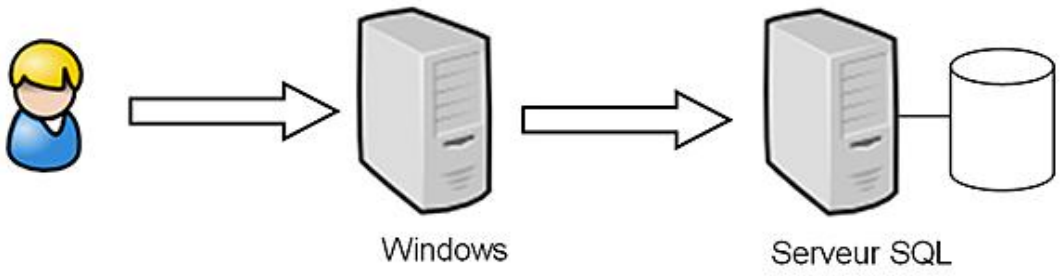
On parlera de connexion au serveur ou de logins.

## 1. Mode de sécurité Windows

Ce type de gestion de la sécurité permet de s'appuyer sur les utilisateurs et les groupes Windows pour le domaine et le poste local. SQL Server utilise la gestion des utilisateurs de Windows (gestion des mots de passe...) et récupère uniquement les noms pour créer des connexions au serveur.

Une fonctionnalité très importante de SQL Server est de pouvoir autoriser des groupes Windows à venir se connecter. La gestion des groupes permet de réaliser une administration plus souple que celle des utilisateurs. La méthode la plus simple pour gérer les connexions est de passer par la création d'un groupe local. Ce groupe local est autorisé à se connecter au serveur SQL et est utilisé par des utilisateurs ou des groupes globaux.

Avec une telle méthode de fonctionnement, comme un utilisateur Windows peut appartenir à plusieurs groupes, il peut posséder plusieurs droits de connexion à SQL Server.




*Authentification Windows*

En mode sécurité Windows, seuls les noms d'utilisateurs sont stockés. La gestion des mots de passe et de l'appartenance aux différents groupes est laissée à Windows. Ce mode de fonctionnement permet de bien discerner les tâches de chacun et de spécialiser SQL Server sur la gestion des données en laissant Windows gérer les utilisateurs, ce qu'il sait bien faire. De plus avec un tel schéma de fonctionnement, il est possible d'appliquer la politique suivante : un utilisateur = un mot de passe. L'accès au serveur SQL est transparent pour les utilisateurs approuvés par Windows.

SQL Server s'appuie sur les groupes auxquels appartient l'utilisateur lors de la connexion au serveur. Si des modifications d'appartenance aux groupes sont effectuées depuis Windows, ces modifications ne seront prises en compte que lors de la prochaine connexion de l'utilisateur au serveur SQL.

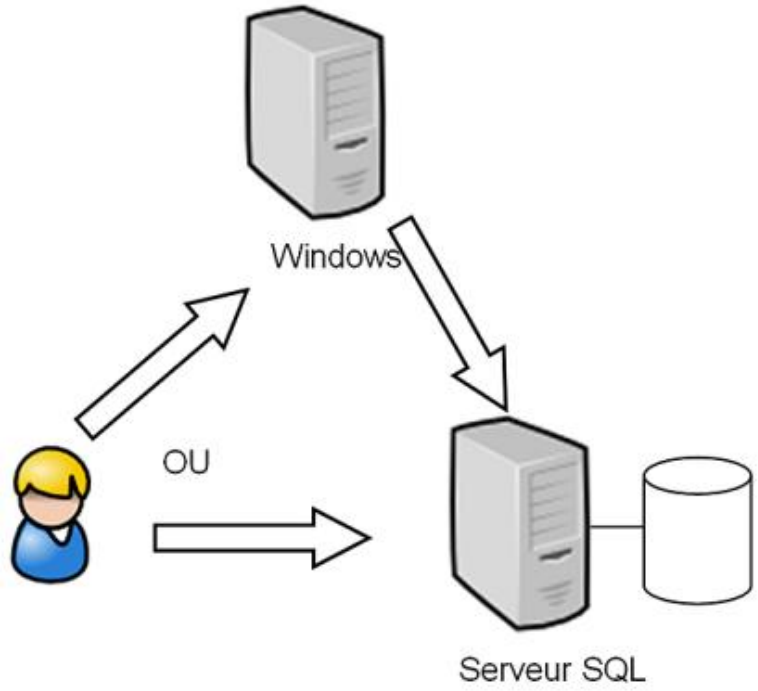
SQL Server s'appuie sur le SID de Windows pour identifier le groupe. Si un groupe est supprimé puis recréé dans Windows, il est important de réaliser la même opération en ce qui concerne la connexion du groupe dans SQL Server.

## 2. Mode de sécurité Mixte

Le mode de sécurité Mixte repose sur une authentification Windows puis sur une authentification SQL Server. C'est ce mode d'authentification qui va être détaillé ici.

### a. Définition

Il s'agit du fonctionnement le plus connu pour la sécurité des serveurs de bases de données. Dans un tel mode de fonctionnement, c'est SQL Server qui se charge de vérifier que l'utilisateur qui demande à se connecter est bien défini puis il se charge également de vérifier le mot de passe.




*Mode de sécurité mixte*

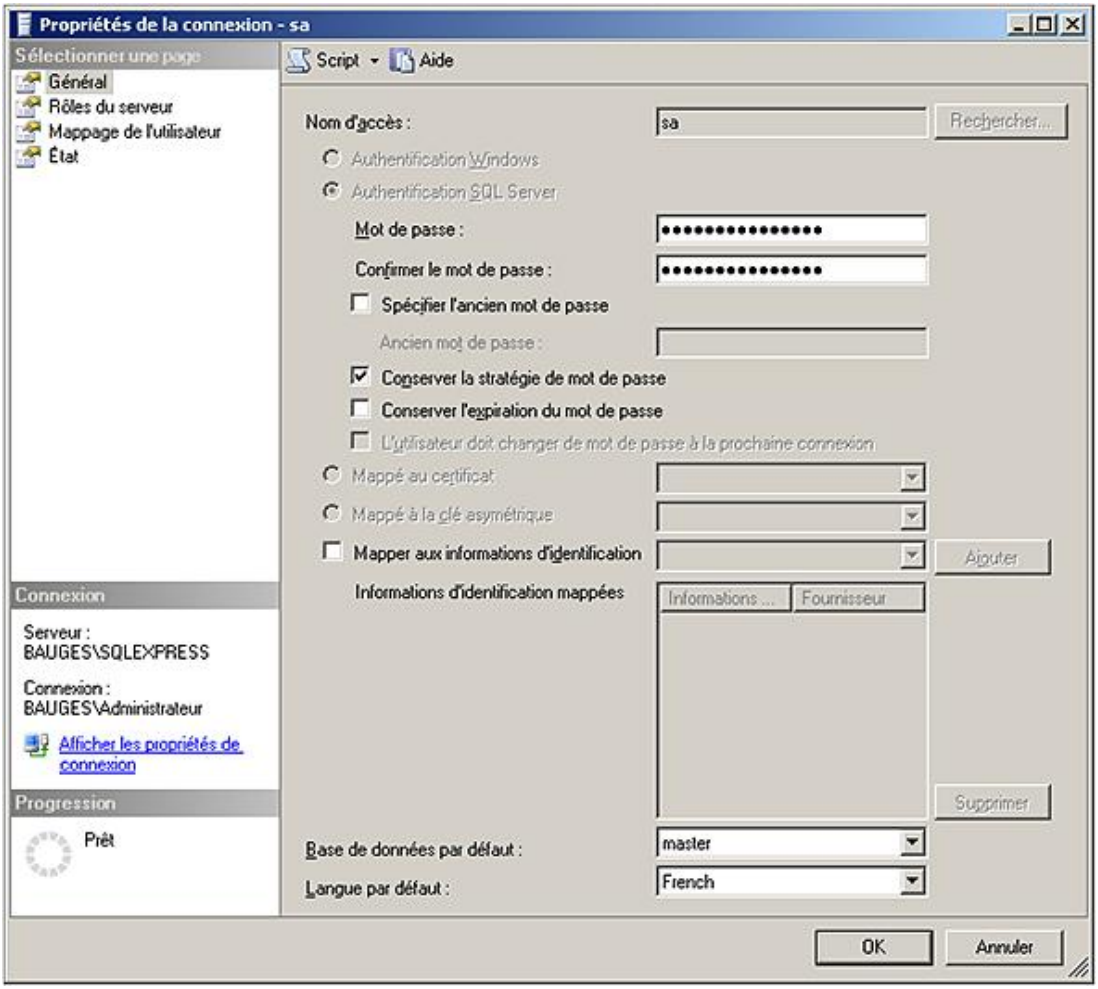
Tous les utilisateurs sont entièrement gérés par SQL Server (nom et mot de passe). Ce type de gestion des connexions est bien adapté pour des clients qui ne s'identifient pas auprès de Windows.

### b. Principe de fonctionnement

Il est trompeur de croire qu'en mode de fonctionnement Mixte seul le serveur se charge de la vérification des noms d'utilisateur et des mots de passe. Dans un premier temps, lorsqu'un utilisateur du réseau tente de se connecter au serveur SQL, un test est fait en utilisant la sécurité Windows. Si l'utilisateur du réseau est approuvé dans SQL Server, ou s'il appartient à un groupe qui est approuvé dans SQL Server, alors l'utilisateur sera connecté au serveur SQL. Dans le cas contraire (échec de la connexion en utilisant la sécurité Windows), SQL Server se charge de demander un nom de connexion et un mot de passe.

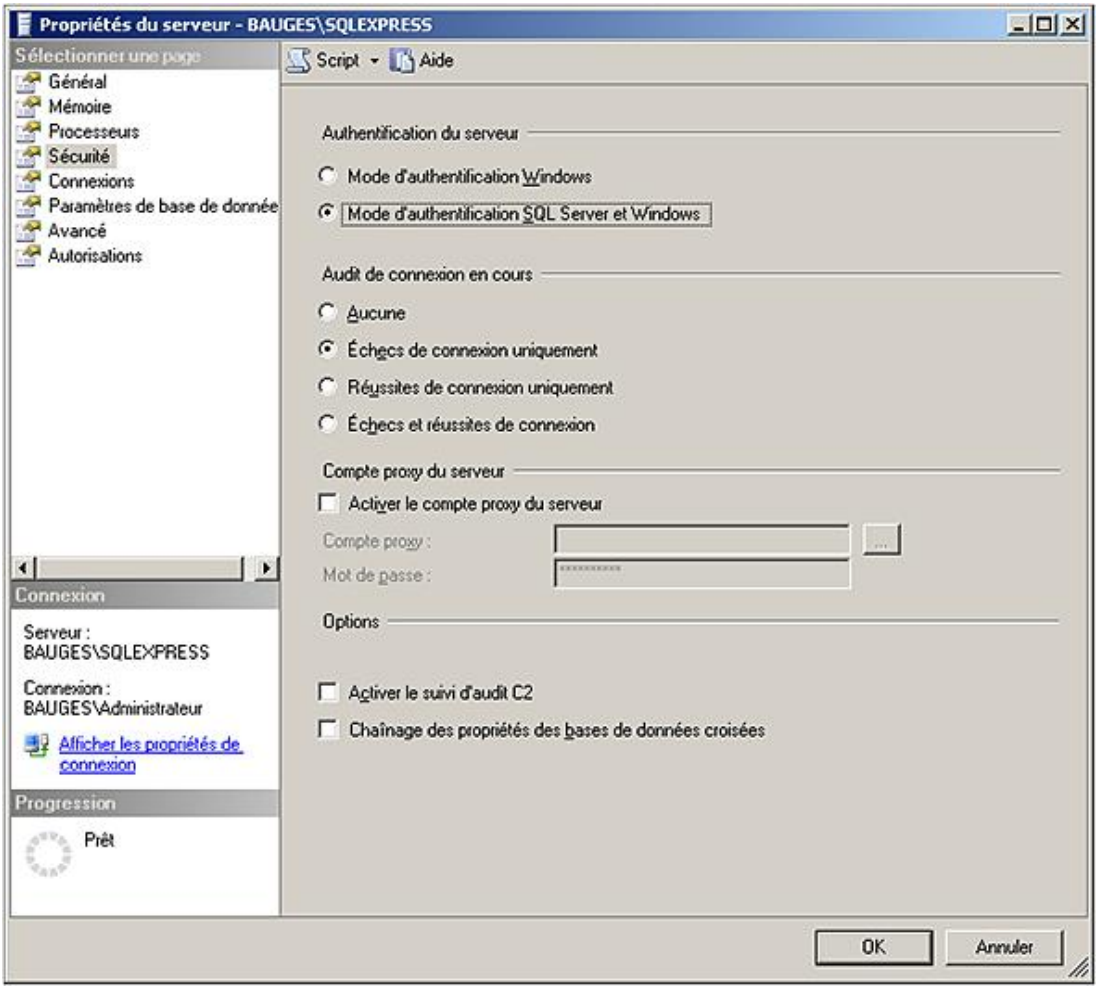
## 3. Base de données par défaut

Après la définition des connexions (logins) au serveur, il est important de définir dans les différentes bases de données utilisateur, des utilisateurs qui correspondent à ces connexions. Les droits d'utilisation à l'intérieur de la base seront attribués aux utilisateurs de la base. Il est important également de définir pour chaque connexion ou login, une base de données par défaut. C'est sur cette base que sera positionné tout utilisateur qui utilise cette connexion. Attention à bien prendre en compte le fait que définir une base de données par défaut ne donne pas de droits d'utilisation sur cette base.




## 4. Comment choisir un mode de sécurité ?

Il est possible de modifier le mode de sécurité utilisé par le serveur SQL directement depuis SQL Server Management Studio en allant modifier les propriétés de l'instance comme ci-après.




Lors de l'installation du serveur SQL, deux connexions sont prédéfinies, en mode Windows, le groupe local des Administrateurs est autorisé à se connecter, et en mode de sécurité SQL Server, l'utilisateur **sa** peut se connecter au serveur. Ces deux utilisateurs possèdent des privilèges d'administrateur du serveur SQL.

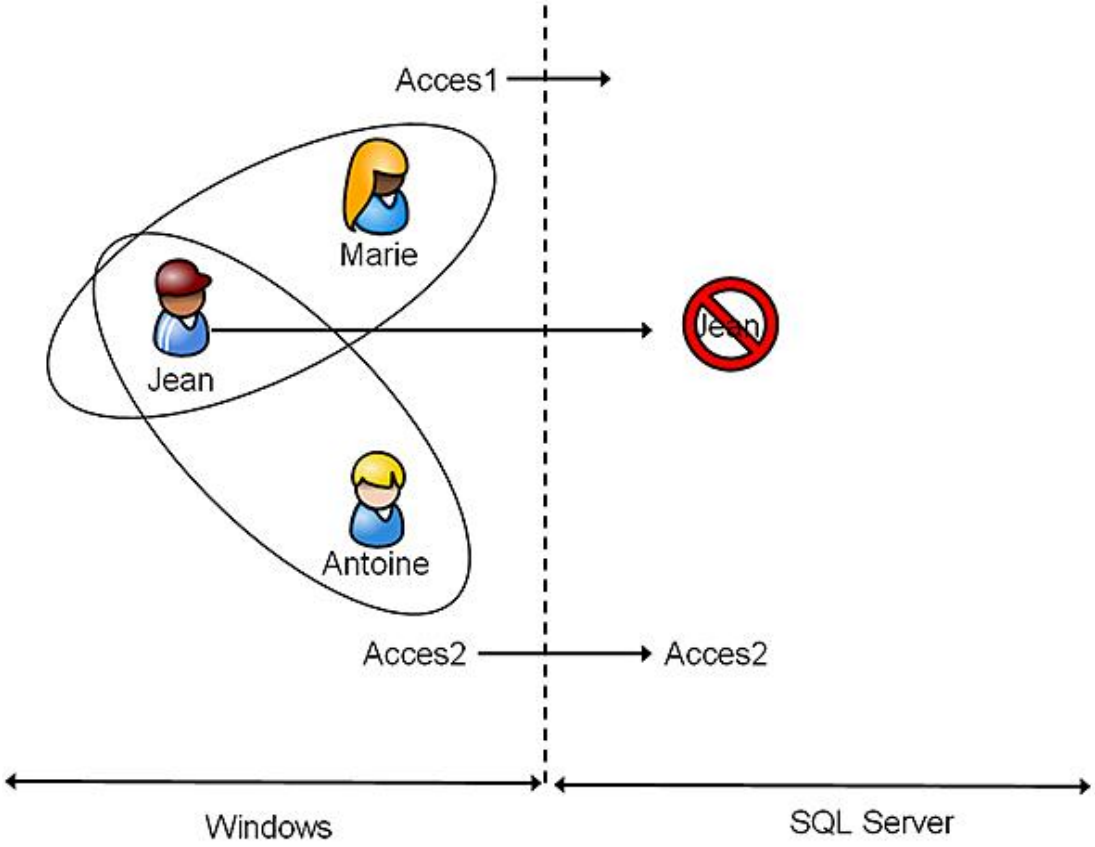
---

Il est recommandé d'utiliser la sécurité Windows qui offre une plus grande souplesse au niveau de la gestion des utilisateurs.

---

## 5. Gérer une connexion à SQL Server

Étant donné qu'un utilisateur Windows peut appartenir à plusieurs groupes, il peut se voir accorder plusieurs fois le droit de connexion au serveur SQL. Ceci peut poser un problème lorsqu'un utilisateur ou un groupe d'utilisateurs ne doit jamais pouvoir se connecter au serveur SQL. Afin de remédier à ce souci, Microsoft fournit, parmi les ordres Transact SQL pour la gestion des connexions, l'ordre DENY qui permet de refuser explicitement un utilisateur ou un groupe Windows. Le DENY constitue un refus explicite et est prioritaire par rapport aux différentes autorisations de connexion.




*Création d'une connexion*

Dans le schéma ci-dessus, il y a trois utilisateurs Windows et deux groupes. Une connexion a été mise en place pour le groupe Acces2, le groupe Acces1 ne possède pas de connexion et l'utilisateur Jean est interdit de connexion.

---

Il faut posséder une permission d'administrateur (**sysadmin**) ou une permission de gestionnaire de sécurité (**securityadmin**) pour pouvoir réaliser les différentes opérations relatives à la gestion des connexions.

---

Les connexions, qu'elles soient de type SQL Server ou bien Windows, doivent être définies au niveau de l'instance SQL Server pour permettre aux utilisateurs de se connecter.

La gestion des connexions peut être réalisée de façon graphique par l'intermédiaire de l'explorateur d'objets depuis SQL Server Management Studio ou bien sous forme de script Transact SQL. Toutes les opérations de gestion des connexions sont réalisables en Transact SQL avec l'aide des instructions CREATE LOGIN, ALTER LOGIN et DROP LOGIN. Chaque solution possède ses avantages et ses inconvénients.

---

Les procédures sp_addlogin et sp_grantlogin ne doivent plus être utilisées. Elles sont encore présentes dans SQL Server 2008 pour assurer la compatibilité des scripts.

---

Les comptes de connexion sont nécessaires pour permettre l'accès au serveur de base de données. Toutefois, ils sont insuffisants pour permettre de travailler sur une base de données. Il faut en effet leur associer une base de données par défaut, c'est-à-dire la base de travail par défaut sur laquelle ils seront positionnés après ouverture de la connexion. Pour accéder à cette base de données, un compte d'utilisateur de base de données doit être défini pour la connexion sur la base de données.

### a. En mode de sécurité Windows

Les noms des groupes ou des utilisateurs devront correspondre à ceux définis dans Windows.

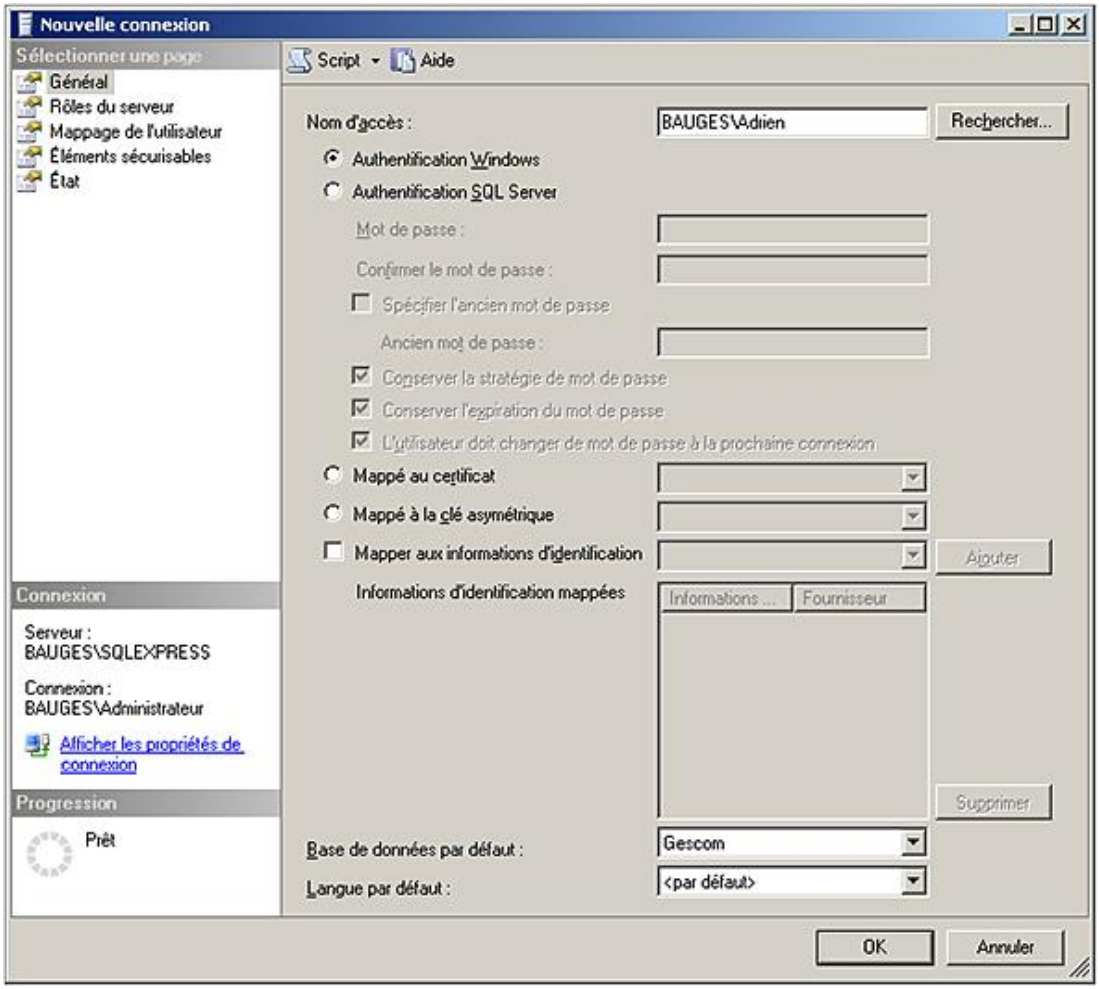
#### SQL Server Management Studio

Il est possible de créer les connexions depuis l'interface graphique de SQL Server Management Studio, en procédant de la sorte :

- Depuis l'explorateur d'objets, se placer sur le nœud **Sécurité-Connexion**.

- Depuis le menu contextuel associé au nœud connexion, faire le choix **nouvelle connexion**.

L'écran suivant apparaît alors. Il ne reste plus qu'à sélectionner le compte d'utilisateur Windows ou bien le groupe qui est autorisé à se connecter.




L'interface graphique de SQL Server Management Studio permet également de modifier simplement un compte de connexion à partir de la fenêtre des propriétés ou bien de supprimer une connexion.

## Transact SQL

```
CREATE LOGIN nom_connexion
FROM WINDOWS
[WITH DEFAULT_DATABASE=baseDeDonnées|DEFAULT_LANGUAGE=langue]
```

`nom_connexion`

Nom de l'utilisateur ou du groupe Windows à ajouter. Il doit être donné sous la forme domaine\utilisateur.

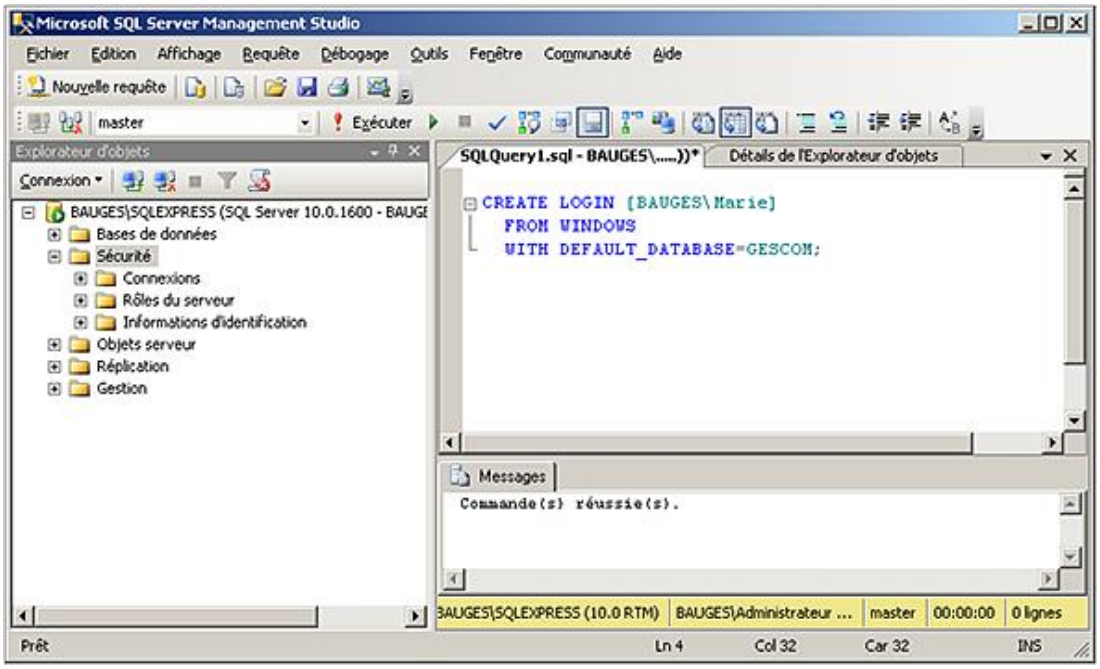
`baseDeDonnées`

Précise la base de données qui sera utilisée par défaut. Si aucune base de données n'est précisée, alors la base de données par défaut est la base Master.

`langue`

Langue de travail par défaut pour cette connexion. Cette option n'est à préciser que si la langue relative à cette connexion est distincte de celle définie par défaut sur le serveur SQL.

*Exemple :*




## b. En mode de sécurité Mixte

Comme pour la sécurité en mode Windows, il existe deux moyens pour gérer les connexions.

Mis à part l'ensemble des règles de gestion des mots de passe, la gestion d'une connexion en mode de sécurité SQL Server est en tout point semblable à celui d'une connexion en mode de sécurité Windows.

Une fois créés, les deux types de connexion sont en tout point identiques. Il n'y a pas une solution qui présente plus de fonctionnalités que l'autre.

Pour les connexions utilisant une sécurité SQL Server, il est possible de définir des règles de gestion des mots de passe.

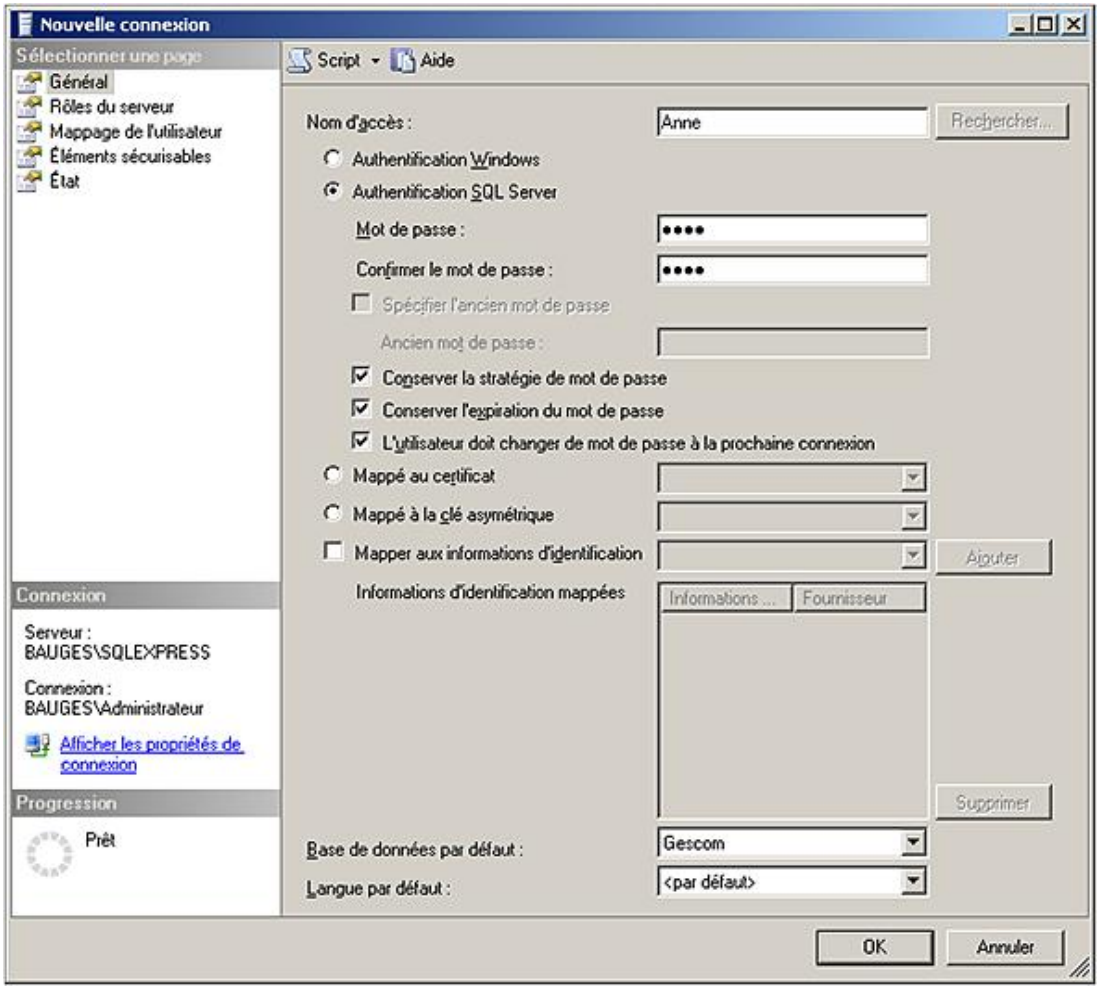
Pour les connexions utilisant une sécurité Windows, c'est le système d'exploitation qui gère les règles de sécurité relatives au mot de passe.

### SQL Server Management Studio

Il est possible de créer les connexions depuis l'interface graphique de SQL Server Management Studio, en procédant de la sorte :

- Depuis l'explorateur d'objets, se placer sur le nœud **Sécurité-Connexion**.
- Depuis le menu contextuel associé au nœud de connexion, faire le choix **nouvelle connexion**.

L'écran suivant apparaît alors. Il ne reste plus qu'à définir le nom de la nouvelle connexion ainsi que le mot de passe associé. C'est également à ce niveau que peut être précisée la politique de gestion des passes à mettre en place. Les règles de gestion des mots de passe sont héritées de celles mise en place sur Windows.




L'interface graphique de SQL Server Management Studio permet également de modifier simplement un compte de connexion à partir de la fenêtre des propriétés ou bien de supprimer une connexion.

### Transact SQL

```
CREATE LOGIN nom_connexion
WITH {PASSWORD='motDePasse' |motDePasseHaché HASHED}[MUST_CHANGED]
[,SID=sid |
,DEFAULT_DATABASE=baseDeDonnées |
,DEFAULT_LANGUAGE=langue |
,CHECK_EXPIRATION={ON | OFF} |
,CHECK_POLICY={ON | OFF}
,[CREDENTIAL=nom_credit]]
```

`nom_connexion`

Nom de la connexion qui sera définie dans SQL Server.

`motDepasse`

Mot de passe associé à la connexion. Ce mot de passe est stocké de façon cryptée dans la base Master et il est vérifié lors de chaque nouvelle connexion au serveur. Il est obligatoire de définir un mot de passe pour chaque connexion.

`motDePasseHaché`

Il s'agit ici de préciser que la chaîne fournie correspond à la version hachée du mot de passe. Ce n'est donc pas le mot de passe tel que l'utilisateur le saisit, mais le mot de passe tel que SQL le stocke qui est fourni.

`MUST_CHANGED`

Avec cette option SQL Server demande à l'utilisateur de saisir un nouveau mot de passe lors de sa première connexion au serveur. Cette option n'est possible que si `CHECK_ESPIRATION` et `CHECK_POLICY` sont activées (positionnées à ON).

`baseDeDonnées`

Précise la base de données qui sera utilisée par défaut. Si aucune base de données n'est précisée alors la base de données par défaut est la base Master.

`langue`

Langue de travail par défaut pour cette connexion. Cette option n'est à préciser que si la langue relative à cette connexion est distincte de celle définie par défaut sur le serveur SQL.

`CHECK_EXPIRATION`

Si cette option est positionnée à ON (OFF par défaut), elle indique que le compte de connexion va suivre la règle relative à l'expiration des mots de passe définie sur le serveur SQL. L'activation de cette option n'est possible que si `CHECK_POLICY` est également activée.

`CHECK_POLICY`

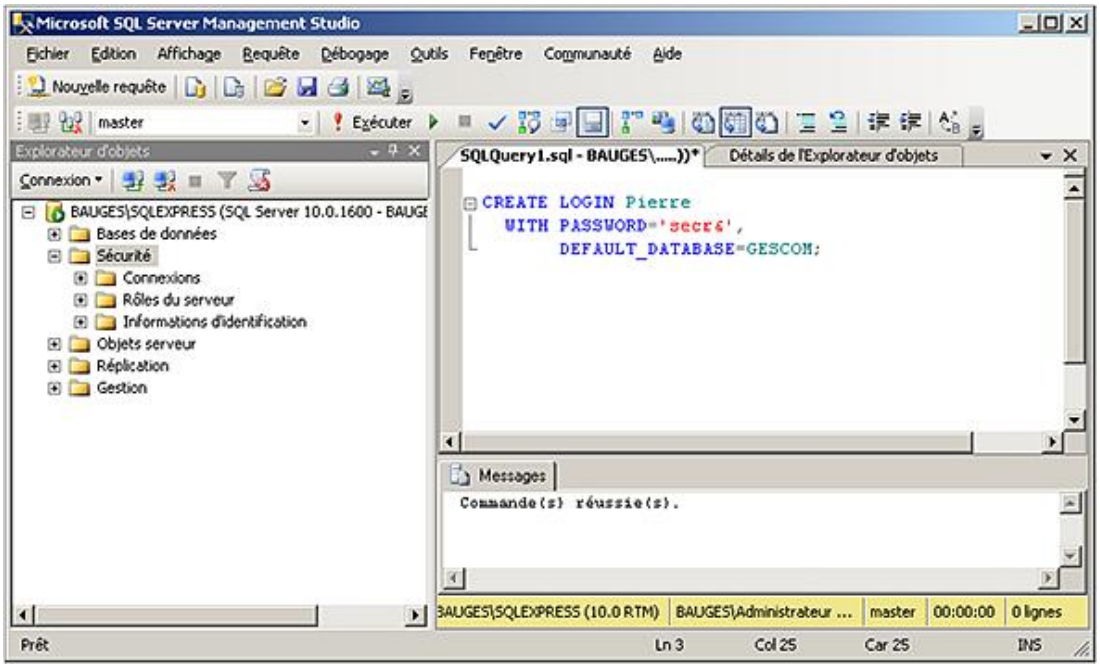
Activée par défaut, cette option permet de répercuter au niveau de SQL Server les règles de gestion des mots de passe définies au niveau de Windows sur le poste qui héberge l'instance SQL Server.

`CREDENTIAL`

Permet de relier la connexion à un credential créé précédemment par l'intermédiaire de l'instruction `CREATE CREDENTIAL`.

*Exemple*

*Dans l'exemple suivant, la connexion PIERRE est créée avec le mot de passe secret. L'utilisateur ne devra pas changer son mot de passe et sa base de données par défaut est configurée.*




## En utilisant un certificat

SQL Server donne la possibilité de créer des connexions au serveur basées sur des certificats (CERTIFICATE). Ces certificats permettent d'identifier de façon sûre un utilisateur en se basant sur différentes informations telles que :

- une clé publique ;

- une identification telle que le nom et l'adresse e-mail ;
- une période de validité.

Pour être tout à fait précis, les certificats de SQL Server sont conformes au standard X.509.

La création d'un certificat dans SQL Server peut s'appuyer sur un certificat défini dans un fichier ou un assembly ou bien demander à SQL Server de générer les clés publiques et privées.

Les certificats de sécurité vont permettre à des applications et/ou des services, d'ouvrir une session sécurisée sur le serveur. Ce type d'ouverture de session par un service est utilisé par le service Service Broker qui utilise un certificat pour poster un message sur une base distante.

## 6. Informations d'identification

Ces objets permettent à des connexions en mode de sécurité SQL Server d'accéder à des ressources externes au serveur de base de données. Les informations d'identification (credentials) vont donc contenir un nom de compte Windows et un mot de passe. Les connexions SQL Server sont reliées à un credential par l'intermédiaire de l'instruction CREATE LOGIN ou bien ALTER LOGIN.

La modification et la suppression sont possibles par l'intermédiaire des instructions ALTER CREDENTIAL et DROP CREDENTIAL.

*Syntaxe*

```
CREATE CREDENTIAL nomDuCredit
WITH IDENTITY = 'identité' [, SECRET = 'secret'];
```

`nomDuCredit`

Permet d'identifier le credential de façon unique au niveau du serveur. Cet identifiant ne peut pas commencer par le caractère #. Les credentials systèmes commencent toujours par les caractères ##.

`identité`

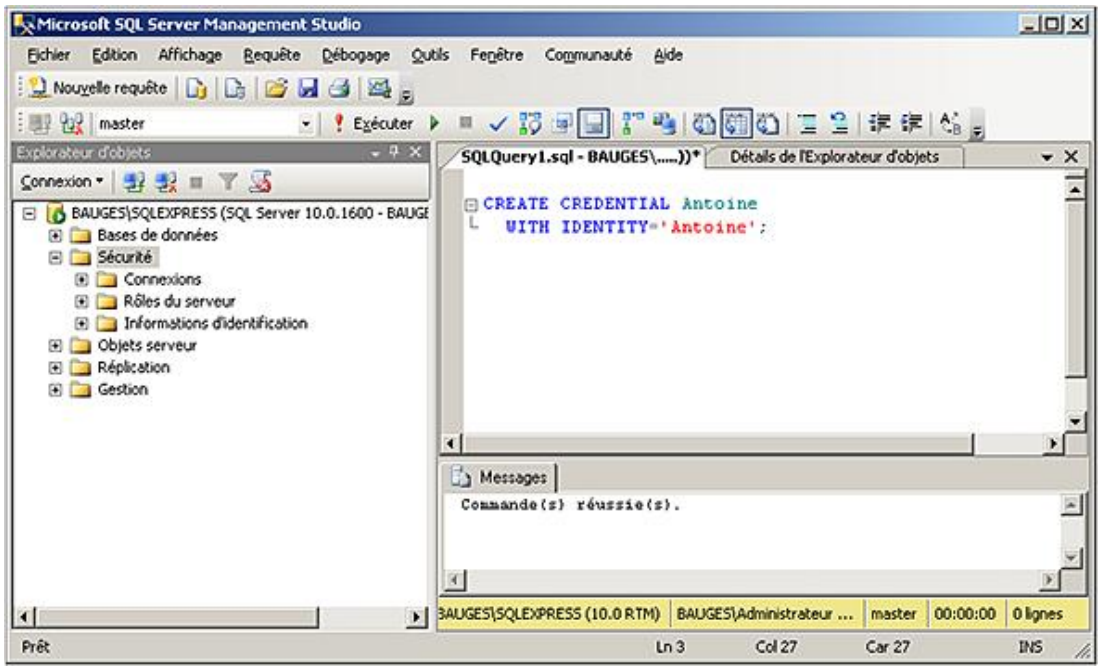
Nom du compte Windows qui sera associé au credential et permettra l'accès à des ressources en dehors de SQL Server.

`secret`

Optionnel, ce paramètre permet de spécifier un mot de passe pour pouvoir accéder à des ressources externes.

*Exemple*

*Un credential est créé et il correspond au compte d'utilisateur Antoine défini au niveau de Windows.*




Il est possible d'avoir toutes les informations relatives aux credentials déjà définis sur le serveur en interrogeant la vue **sys.credential**.

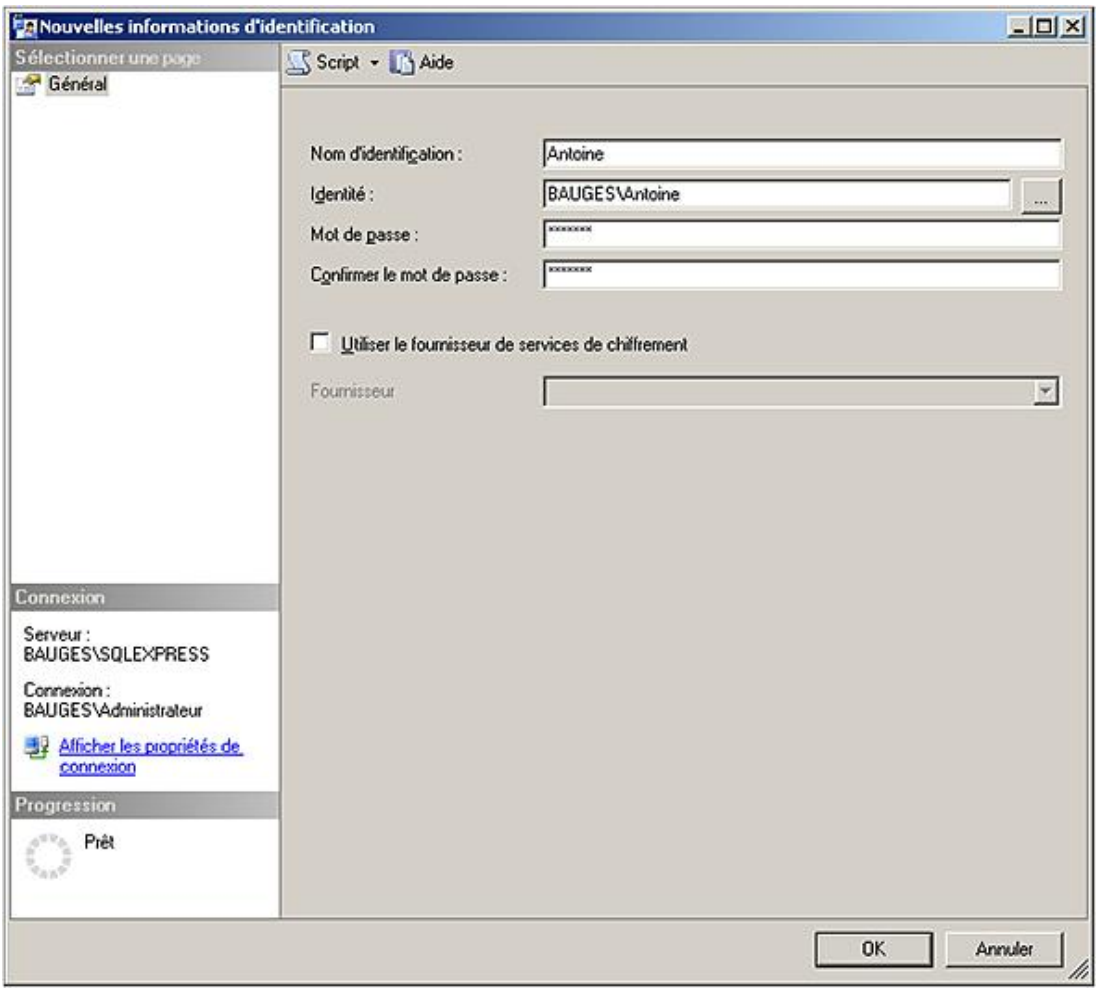
## SQL Server Management Studio

La gestion d'un credential de façon graphique est possible en procédant de la façon suivante :

- Depuis l'explorateur d'objets, se positionner sur le nœud **Securité - Informations d'identification**.
- Depuis le menu contextuel associé au nœud **Informations d'identification**, demander la création d'un credential en choisissant l'option **Nouvelles informations d'identification...**

*Exemple*

*Fenêtre de création d'un nouveau credential*

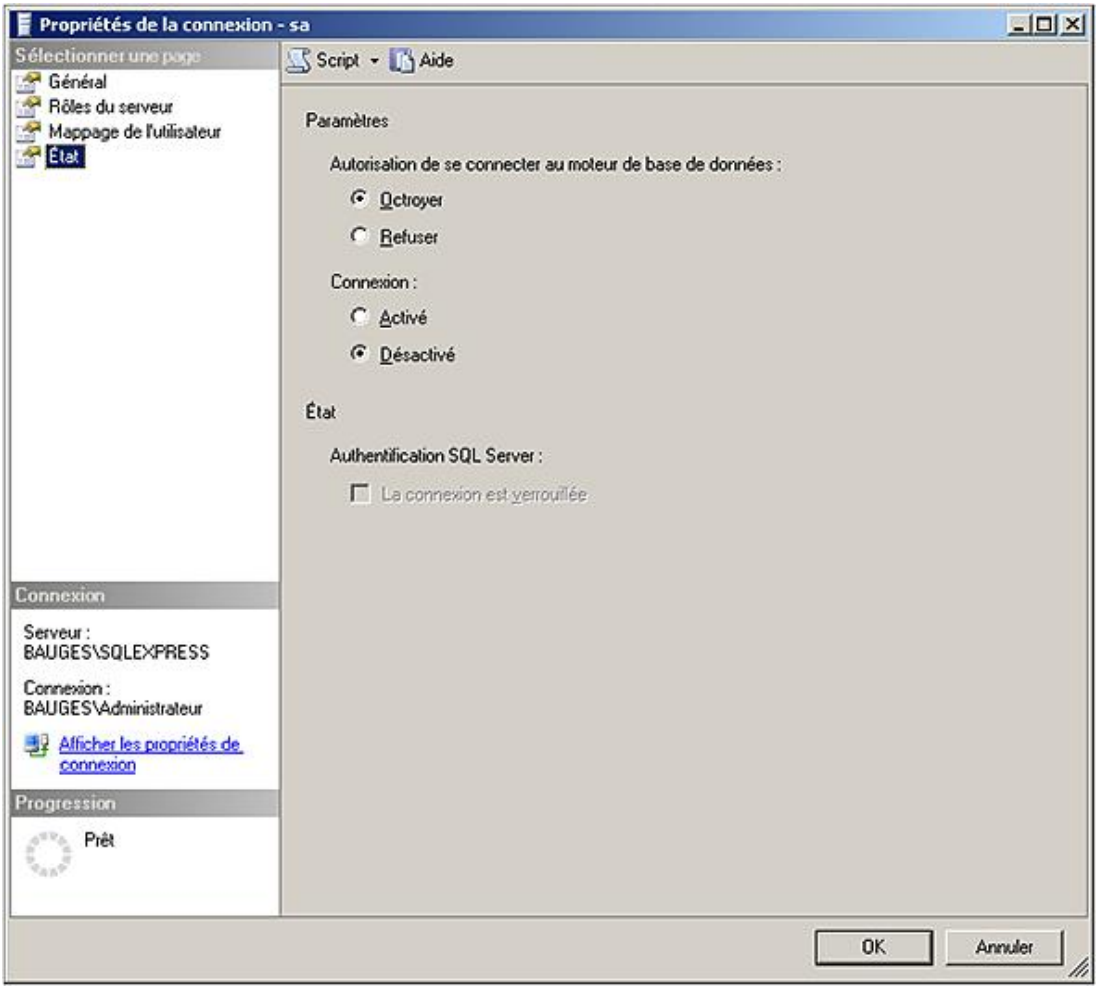



## 7. Activer et désactiver une connexion

Sur une connexion définie, il est possible de la désactiver afin de désactiver temporairement son utilisation. Si toutes les informations et propriétés de la connexion sont conservées, il n'est pas possible d'utiliser la connexion désactivée pour se connecter à SQL Server. La désactivation de connexion est utile pour renforcer la sécurité sur des comptes de connexion qui, bien que définis, ne sont pas utilisés ou bien lorsque l'administrateur définit au préalable des connexions. Il ne lui restera plus qu'à activer la ou les connexions lorsque le besoin sera présent.

### SQL Server Management Studio

Depuis la fenêtre des propriétés de la connexion sur la page **État**, il est possible d'activer, de désactiver la connexion.




## Transact SQL

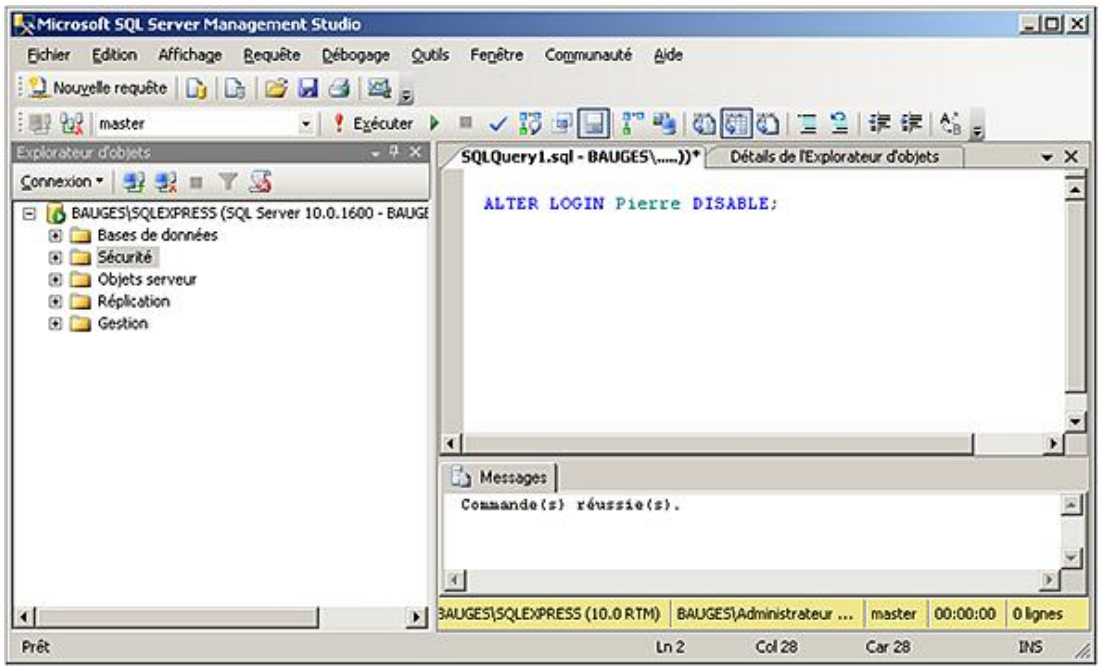
L'activation et la désactivation d'une connexion s'effectuent par l'intermédiaire de l'instruction ALTER LOGIN.

*Syntaxe*

```
ALTER LOGIN nomConnexion {ENABLE|DISABLE}[;]
```

*Exemple*

*La connexion Pierre est désactivée.*



# Gestion des utilisateurs de base de données

Après la définition des connexions (login) au niveau du serveur, il est nécessaire de définir des utilisateurs dans les différentes bases de données.

C'est au niveau des utilisateurs de base de données que seront attribués les droits d'utilisation des objets définis dans la base de données. Lors de la définition d'une connexion, la base de données par défaut permet de positionner le compte de connexion sur une base de données pour commencer à travailler. Cependant, la connexion ne pourra réellement travailler sur la base que s'il existe un compte d'utilisateur défini au niveau base et associé à la connexion. C'est un point de passage obligatoire, sauf si la connexion s'est vue attribuée des privilèges de haut niveau.

---

Si aucune base de données par défaut n'est définie au niveau de la connexion, alors c'est la base Master qui est considérée comme base par défaut, ce qui n'est pas un bon choix.

---

Les utilisateurs de base de données sont associés à une connexion au niveau du serveur. Cependant, certains utilisateurs tels que **guest**, **sys** et **INFORMATION_SCHEMA** ne sont mappés à aucune connexion.

Si un utilisateur dispose d'une connexion à SQL Server mais s'il n'existe pas d'utilisateur de base de données lui permettant d'intervenir sur les bases, l'utilisateur ne peut réaliser que quelques opérations très limitées :

- sélectionner les informations contenues dans les tables système et exécuter certaines procédures stockées,
- accéder à toutes les bases de données utilisateur munies d'un compte d'utilisateur **guest**,
- exécuter les instructions qui ne nécessitent pas d'autorisation telles que la fonction **PRINT**.

Il existe deux types de droits : les droits d'utilisation des objets définis dans une base de données et les droits d'exécuter des instructions SQL qui vont rajouter de nouveaux objets à l'intérieur de la base.

Les utilisateurs de base de données correspondent à une connexion. Ce sont les utilisateurs de base de données qui reçoivent les différents droits qui permettent d'utiliser les bases.

Pour être capable de gérer les accès à la base, il faut posséder des droits de propriétaire de base (**db_owner**) ou de gestionnaire des accès (**db_accessadmin**).

---

Les utilisateurs de base de données sont stockés dans la table système **sysusers** de la base de données sur laquelle l'utilisateur est défini.

---

---

À sa création un utilisateur de base de données ne dispose d'aucun privilège. Il est nécessaire d'accorder tous les droits dont l'utilisateur a besoin.

---

L'utilisateur **guest** (invité) est un utilisateur particulier qui peut être défini sur les bases de données utilisateur. Ce nom d'utilisateur sera utilisé par les connexions au serveur qui ne sont pas mappées avec un utilisateur de base de données. La gestion de cet utilisateur est la même que celle d'un utilisateur quelconque de la base de données.

## 1. Création

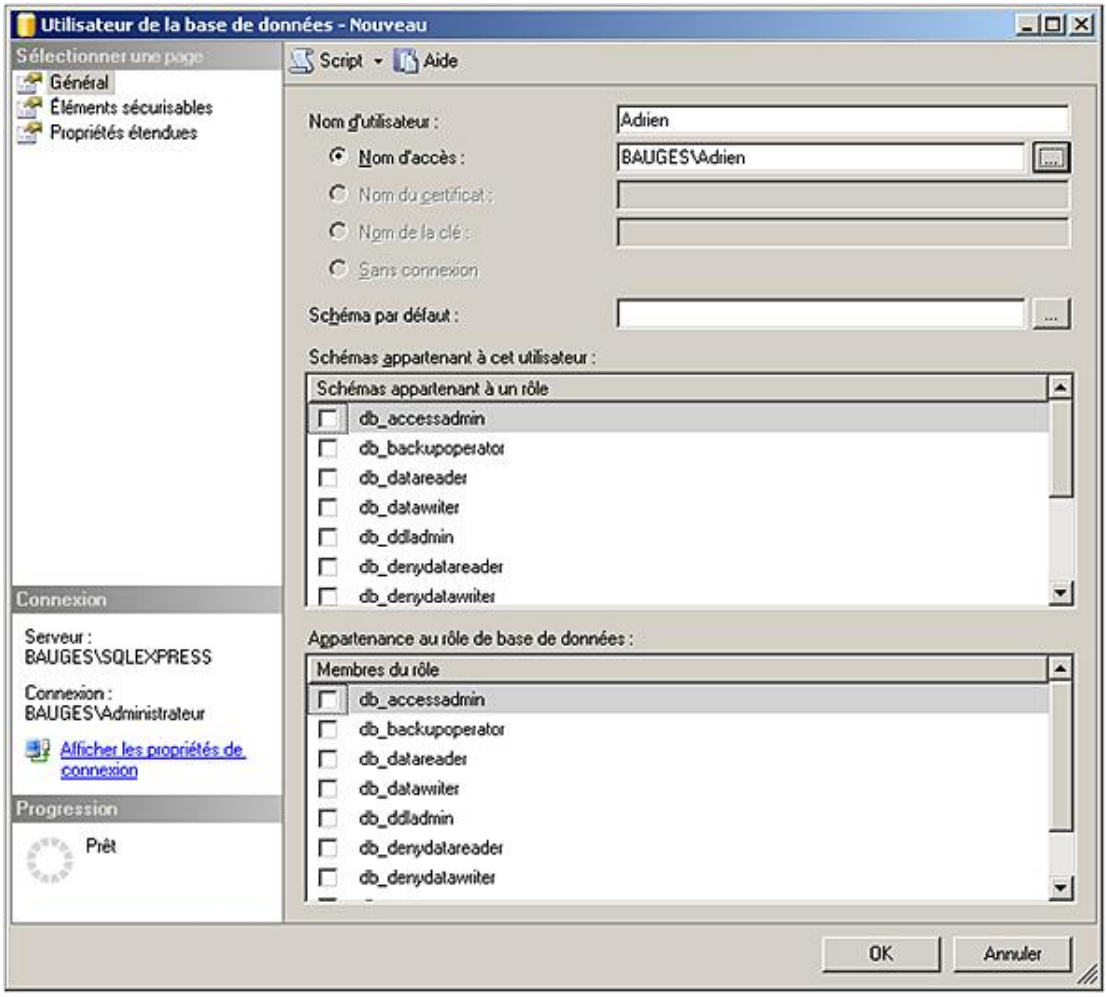
La création d'un utilisateur de base de données va permettre de lier une connexion avec un utilisateur, et donc autoriser l'utilisation de cette base.

### SQL Server Management Studio

Pour créer un utilisateur de base de données, il faut se positionner sur la base de données concernée par l'ajout depuis l'explorateur d'objets et réaliser la manipulation suivante :

- Développer le nœud **Sécurité** et se positionner sur le nœud **Utilisateurs**.
- Depuis le menu contextuel associé au nœud **Utilisateurs**, sélectionner **Nouvel utilisateur**.
- Sélectionner la connexion associée à l'utilisateur, puis définir le nom de ce dernier.

- Enfin, il est possible de préciser les rôles de base de données accordés à cet utilisateur.




Dans l'exemple précédent, l'utilisateur Adrien est créé dans la base de données GESCOM. Cet utilisateur est mappé à la connexion Adrien.

## Transact SQL

C'est par l'intermédiaire de l'instruction CREATE USER qu'il est possible de définir des comptes utilisateurs au niveau de la base de données.

*Syntaxe*

```
CREATE USER nomUtilisateur
[ FOR {LOGIN nomConnexion |
       CERTIFICATE nomCertificat|
       ASYMMETRIC KEY nomCléAsymétrique}]
[ WITH DEFAULT_SCHEMA = nomSchema ]
```

`nomUtilisateur`

Nom du nouvel utilisateur de base de données.

`nomConnexion`

Nom de la connexion à laquelle l'utilisateur de Base de données est associé. Si aucun nom de connexion n'est précisé, alors SQL Server résout le problème de la façon suivante :

- 1 : SQL Server tente d'associer le compte d'utilisateur de base de données à une connexion qui porte le même nom.

- 2 : si l'étape précédente échoue, si le nom d'utilisateur est **guest** et s'il n'existe pas encore de compte de **guest** sur la base de données, alors le compte d'utilisateur **guest** (invité) est créé.

- 3 : dans tous les autres cas l'instruction échoue.

`nomCertificat`

Nom du certificat à associer à l'utilisateur de base de données.

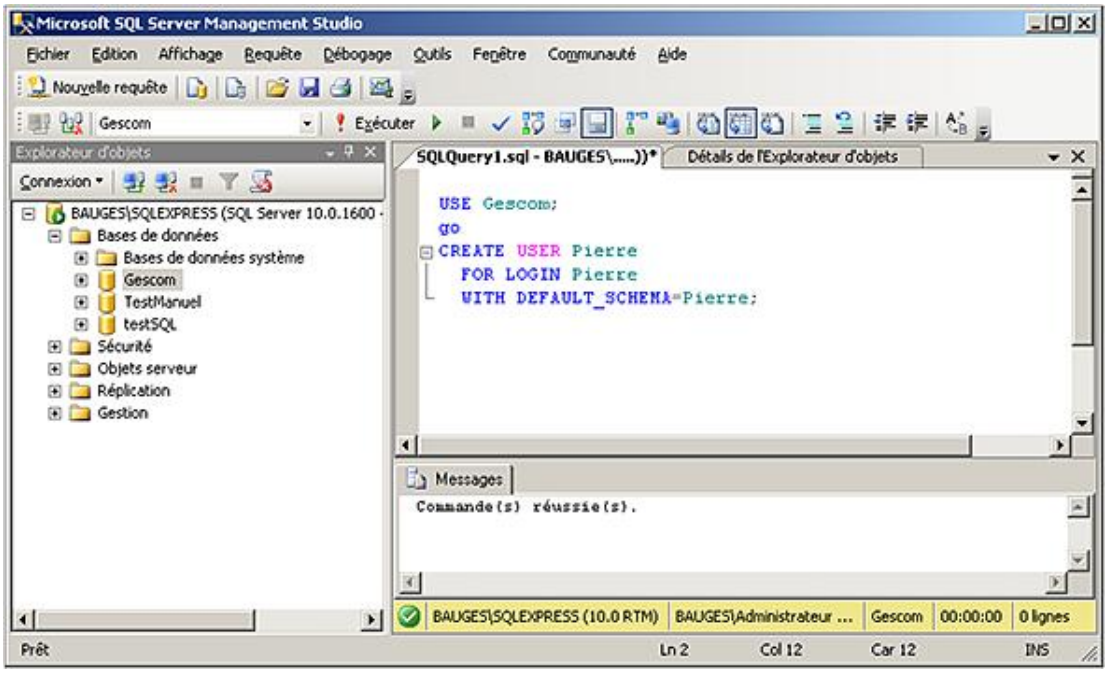
`nomCléAsymétrique`

Nom de la clé asymétrique associée à l'utilisateur de base de données.

`nomSchema`

Nom du schéma associé à cet utilisateur de base de données. Il est possible d'associer plusieurs utilisateurs au même schéma.

*Exemple*




Les procédures sp_grantdbaccess et sp_adduser sont maintenues dans SQL Server uniquement pour des raisons de compatibilité. Il est recommandé de ne plus l'utiliser.

## 2. Information

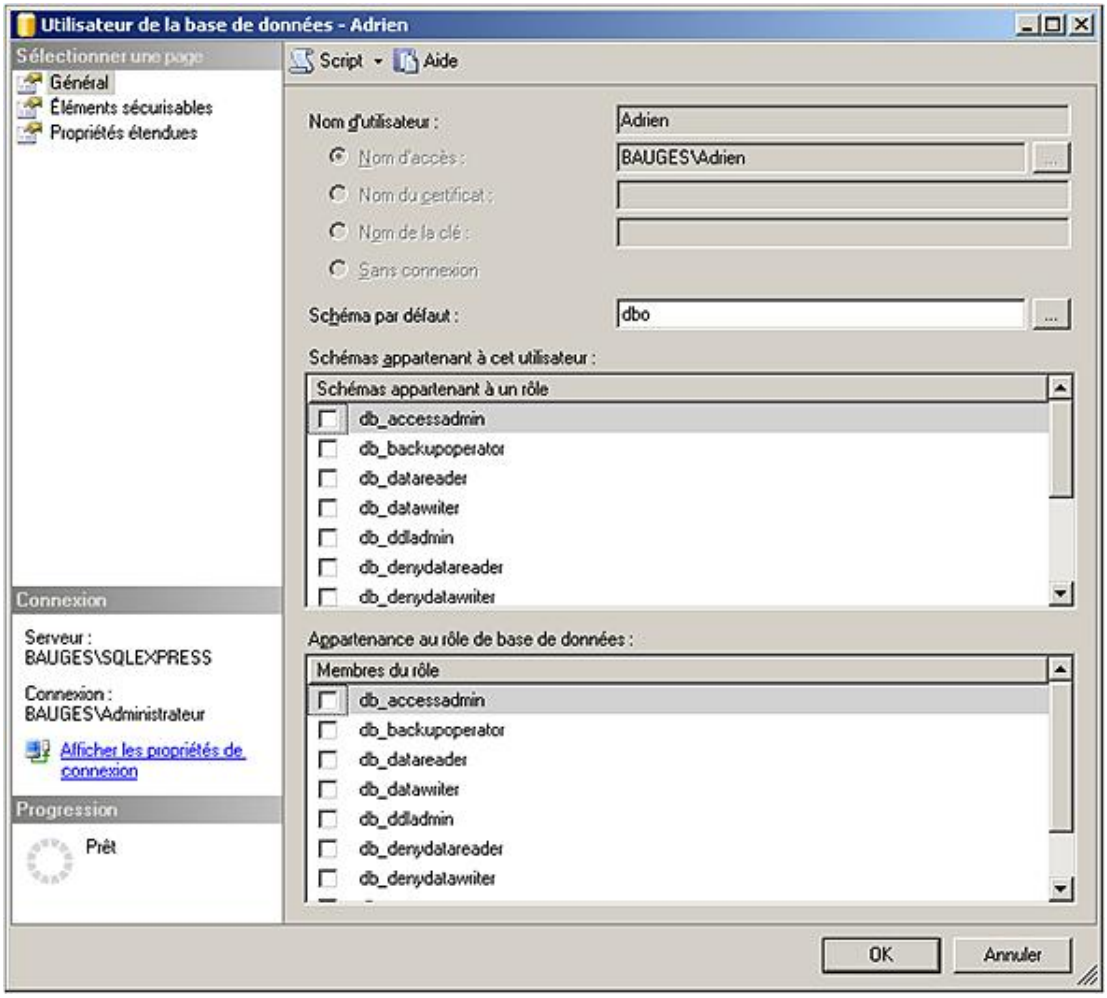
### SQL Server Management Studio

Pour obtenir l'ensemble des informations relatives à un utilisateur de base de données, il faut procéder de la façon suivante :

- Depuis l'explorateur d'objets, se positionner sur la base de données.

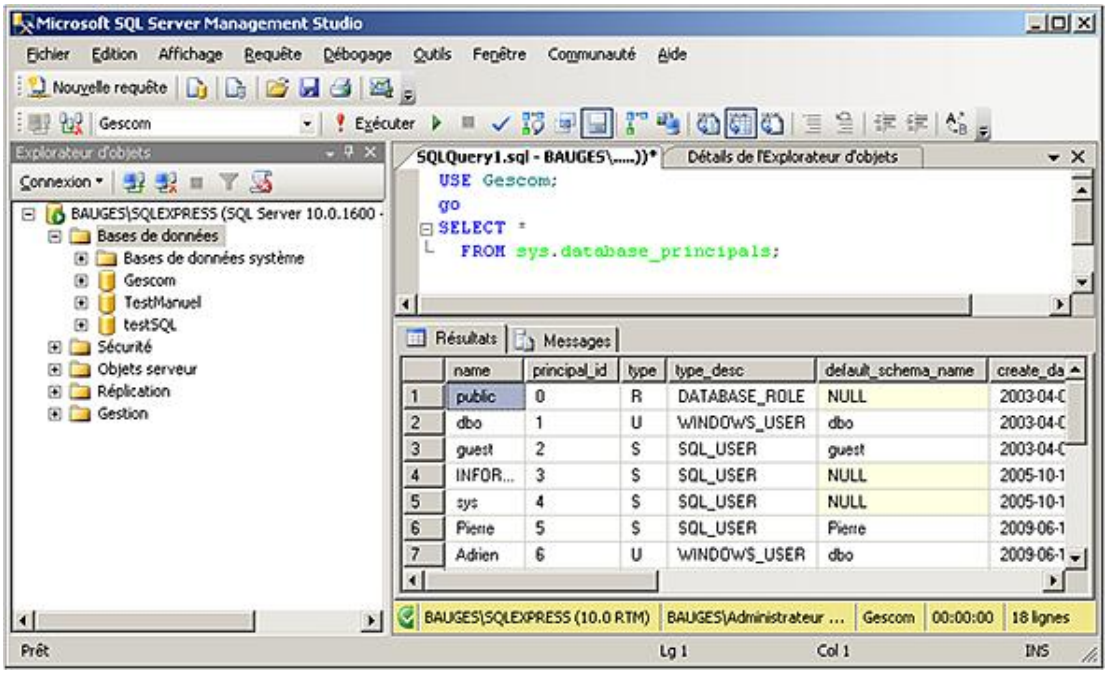
- Développer les nœuds **Sécurité - Utilisateurs**.

- Se positionner sur l'utilisateur pour lequel on souhaite obtenir les informations et demander l'affichage des propriétés à partir du menu contextuel associé.




## Transact SQL

Il est possible d'obtenir l'ensemble des informations relatives aux utilisateurs de base de données en interrogeant la vue **sys.database_principals**.



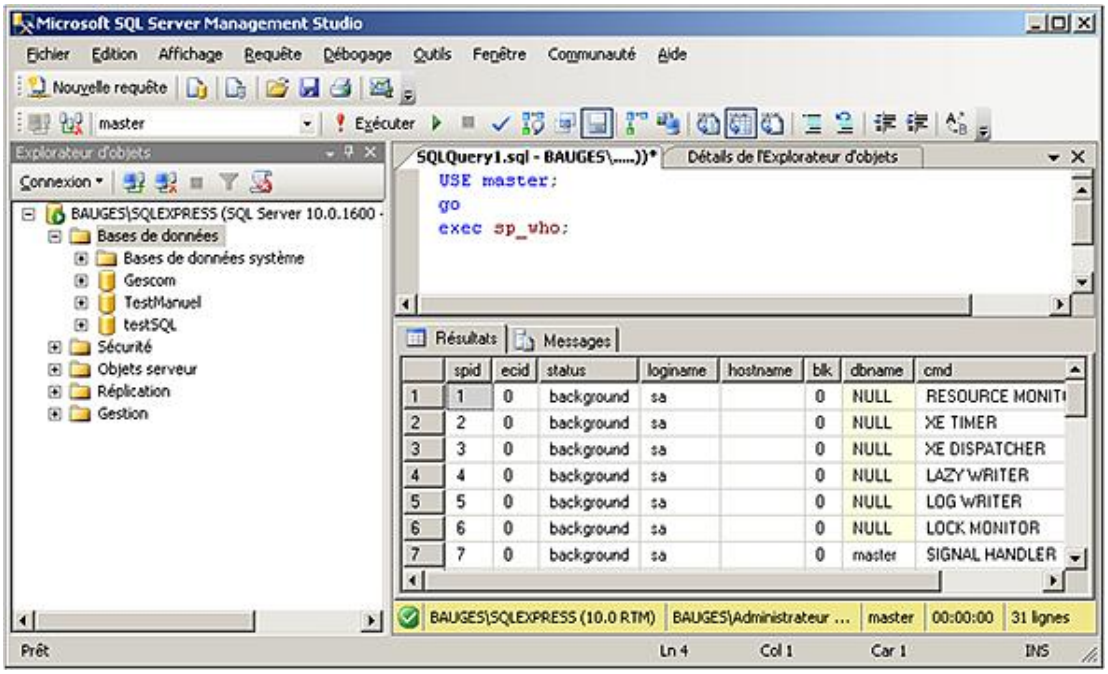
La procédure stockée **sp_helpuser** est maintenue pour des raisons de compatibilité mais il faut lui préférer la vue **sys.database_principals**.

---

La procédure **sp_who** permet de connaître les utilisateurs actuellement connectés et les processus en cours.

Pour chaque connexion, la procédure **sp_who** permet, entre autres, de connaître :

- le nom de connexion utilisé (loginame) ;
- le nom de l'ordinateur à partir duquel la connexion est établie (hostname) ;
- le nom de la base de données courante (dbname).



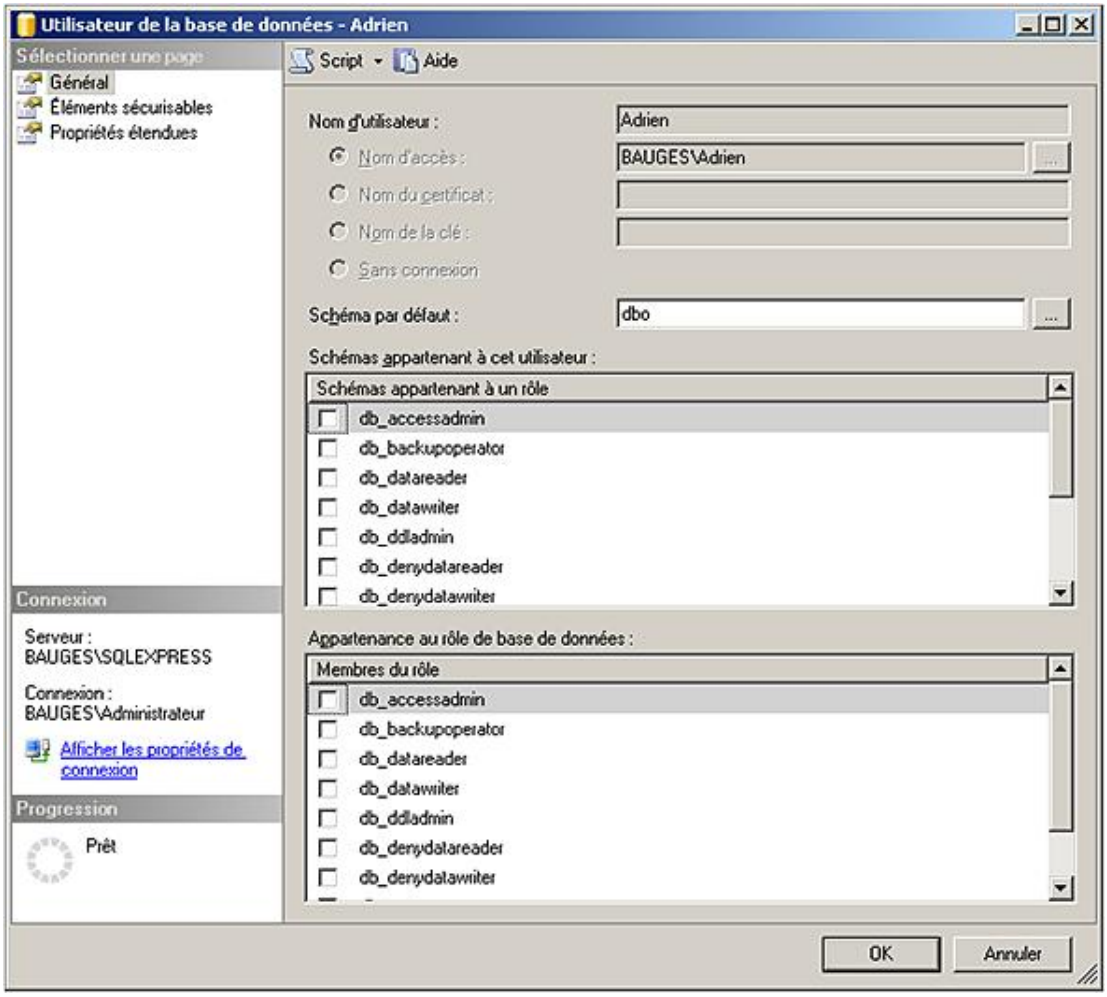

## 3. Modification

Il est possible de modifier un utilisateur de base de données afin de modifier son nom ou bien le schéma associé à cet utilisateur.

### SQL Server Management Studio

Il faut se positionner depuis l'explorateur d'objets sur la base de données concernée par la modification du compte d'utilisateur et réaliser les manipulations suivantes :

- Développer le nœud **Sécurité** puis **Utilisateurs**.
- Se positionner sur le compte d'utilisateur à modifier.
- Afficher la boîte de propriétés depuis l'option **Propriétés** disponible dans le menu contextuel associé à l'utilisateur.



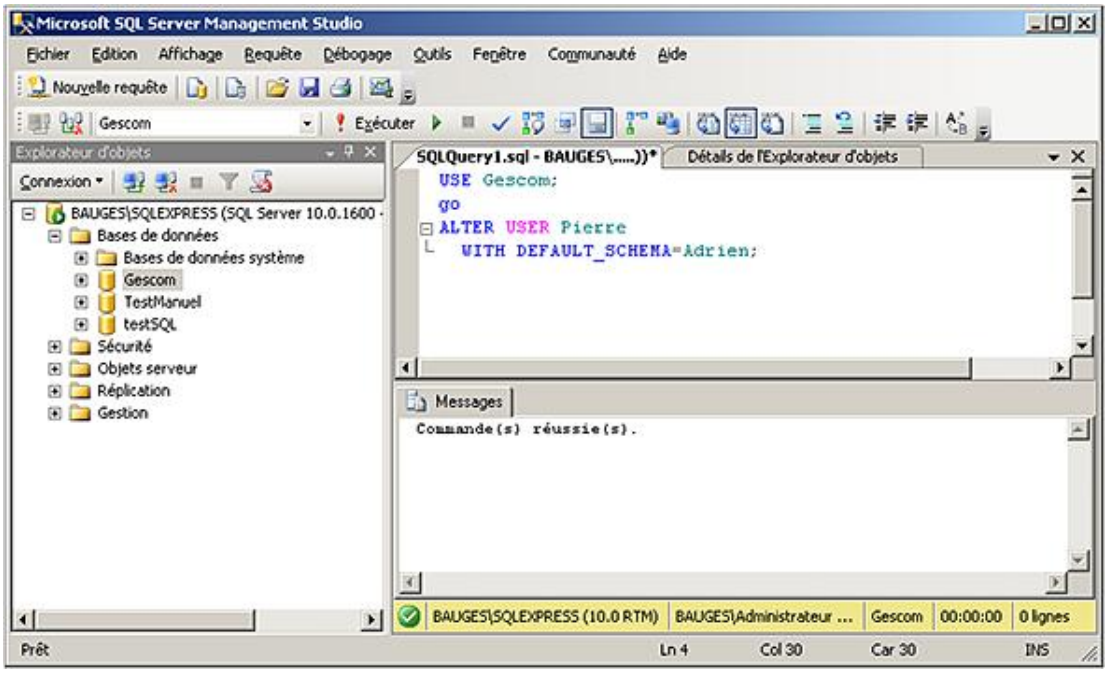

## Transact SQL

L'instruction ALTER USER permet de réaliser cette opération.

*Syntaxe :*

```
ALTER USER nomUtilisateur
WITH NAME=nouveauNomUtilisateur,
     DEFAULT_SCHEMA = nomNouveauSchema
```

*Exemple*




## 4. Suppression

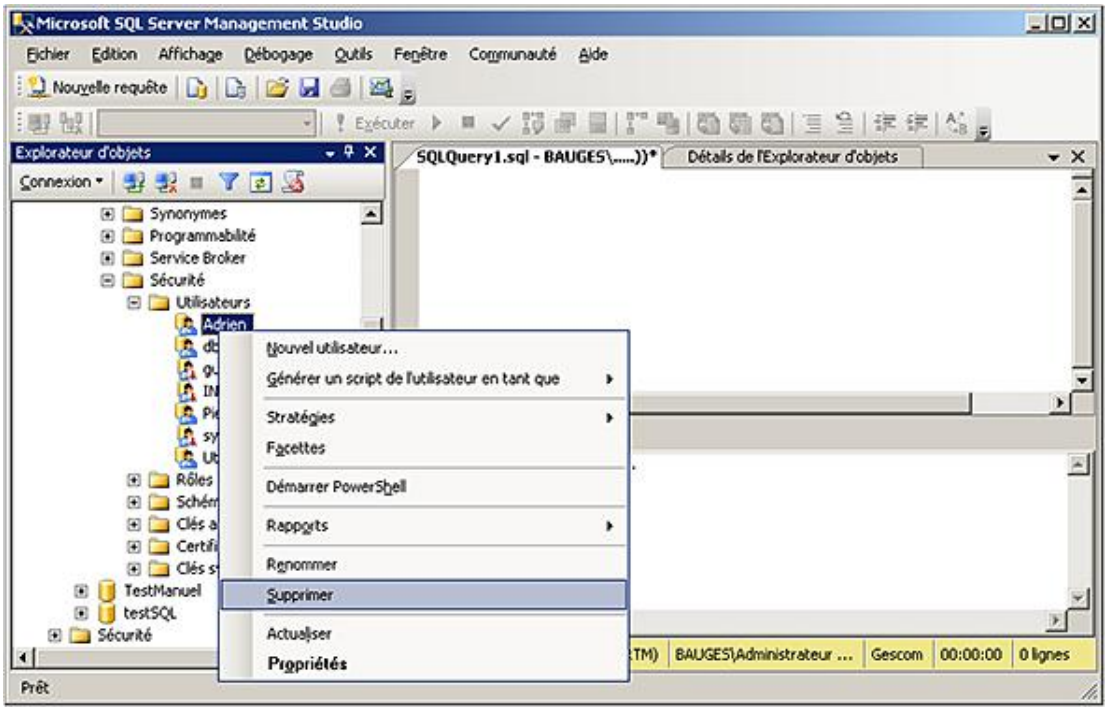
Compte tenu du fait que les utilisateurs ne possèdent pas d'objets, leur suppression est toujours possible et sans aucun risque de perte de données. La suppression d'un utilisateur n'entraîne pas la suppression du schéma associé à l'utilisateur.

### SQL Server Management Studio

Il faut se positionner depuis l'explorateur d'objets sur la base de données concernée par la suppression du compte d'utilisateur et réaliser les manipulations suivantes :

- Développer le nœud **Sécurité** puis **Utilisateurs**.
- Se positionner sur le compte d'utilisateur à supprimer.
- Sélectionner le choix **Supprimer** depuis le menu contextuel associé au compte d'utilisateur.

La suppression n'est pas possible sur les comptes d'utilisateur prédéfinis à savoir : dbo, guest, INFORMATION_SCHEMA et sys.




## Transact SQL

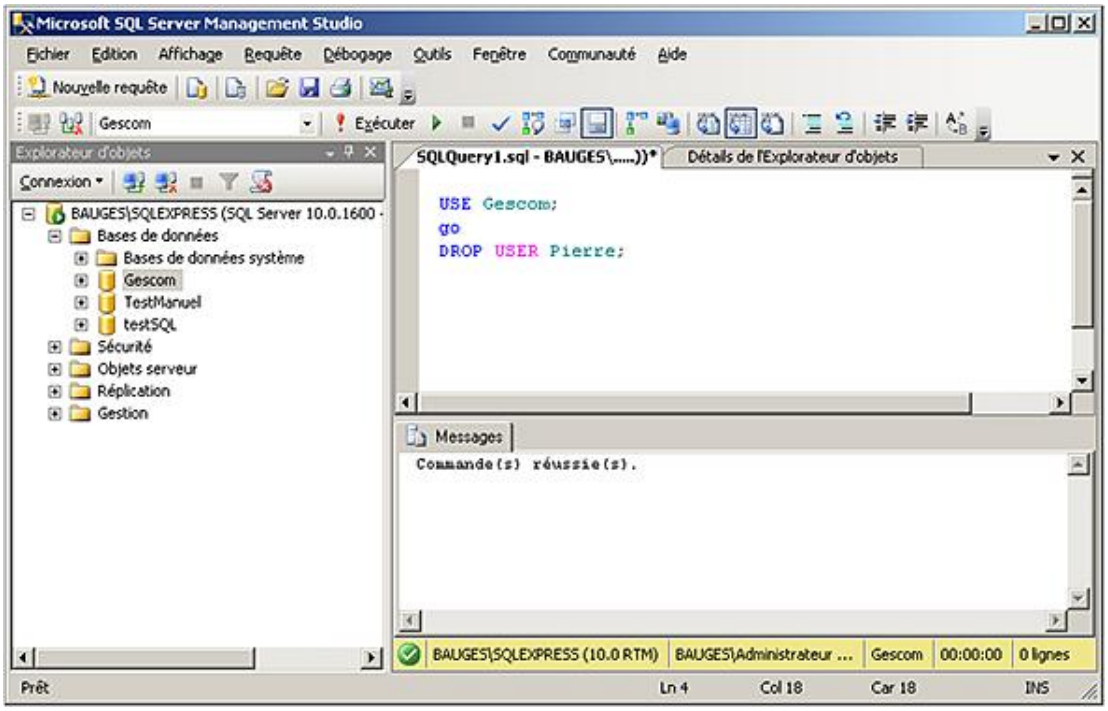
L'instruction DROP USER permet de réaliser cette opération.

Les procédures **sp_dropuser** et **sp_revokedbaccess** sont maintenues dans SQL Server uniquement pour des raisons de compatibilité.

*Syntaxe*

```
DROP USER nomUtilisateur
```

*Exemple :*



# Gestion des schémas

L'objectif des schémas est de dissocier les utilisateurs de base de données des objets qu'ils vont être amenés à créer. Toutefois, les objets ne sont pas laissés tels quels, ils sont regroupés logiquement en schéma. Il est ainsi possible de définir un schéma comme un ensemble logique d'objets à l'intérieur d'une base de données.

Les schémas facilitent le partage d'information entre plusieurs utilisateurs sans pour autant perdre au niveau de la sécurité. Par exemple, si plusieurs développeurs travaillent ensemble sur un même projet, ils vont tous se connecter en utilisant leur propre connexion et utilisateur de base de données, ce qui ne les empêche pas de travailler sur le même schéma et de partager ainsi les tables, vues, procédures, fonctions... qui sont définies sur la base dans le cadre du projet.

Les schémas permettent une gestion plus aisée des privilèges d'utilisation des objets.

Le schéma est associé à un utilisateur de base de données lors de la création ou de la modification de l'utilisateur de la base de données. Si aucun nom de schéma n'est précisé, alors l'utilisateur de base de données va travailler par défaut sur le schéma dbo.

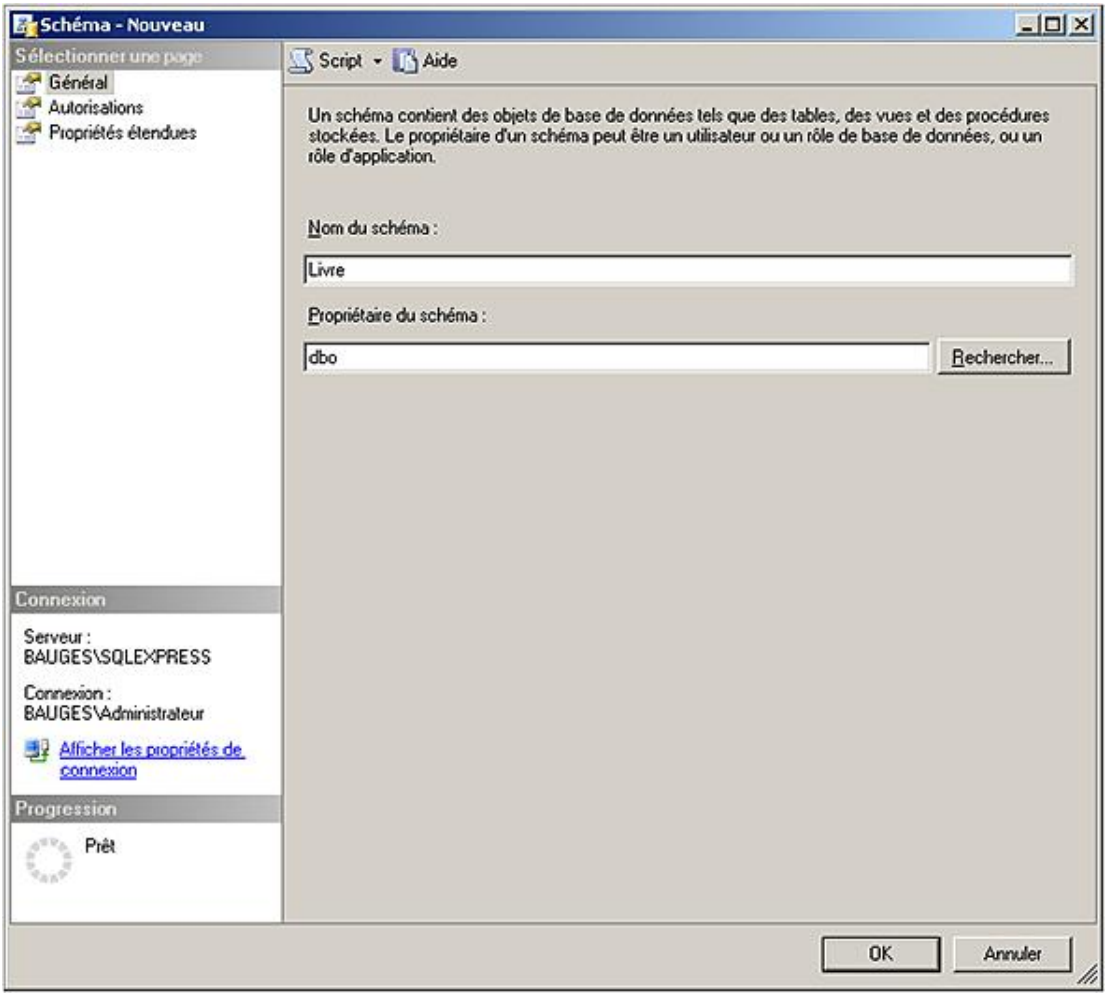
Pour accéder à des objets situés en dehors de son schéma par défaut, un utilisateur de base de données doit utiliser le nom complet d'un objet c'est-à-dire nomSchema.nomObjet. Lors de l'utilisation d'un nom court (simplement le nom de l'objet sans le préfixer par le nom du schéma), SQL Server recherche l'existence de l'objet uniquement dans le schéma courant de l'utilisateur.

## 1. Création

### a. SQL Server Management Studio

Pour créer un schéma de base de données, il faut se positionner sur la base de données concernée par l'ajout et depuis l'explorateur d'objets réaliser la manipulation suivante :

- Développer le nœud **Sécurité** et se positionner sur le nœud **Schémas**.
- Depuis le menu contextuel associé au nœud **Schémas**, sélectionner **Nouveau schéma**.



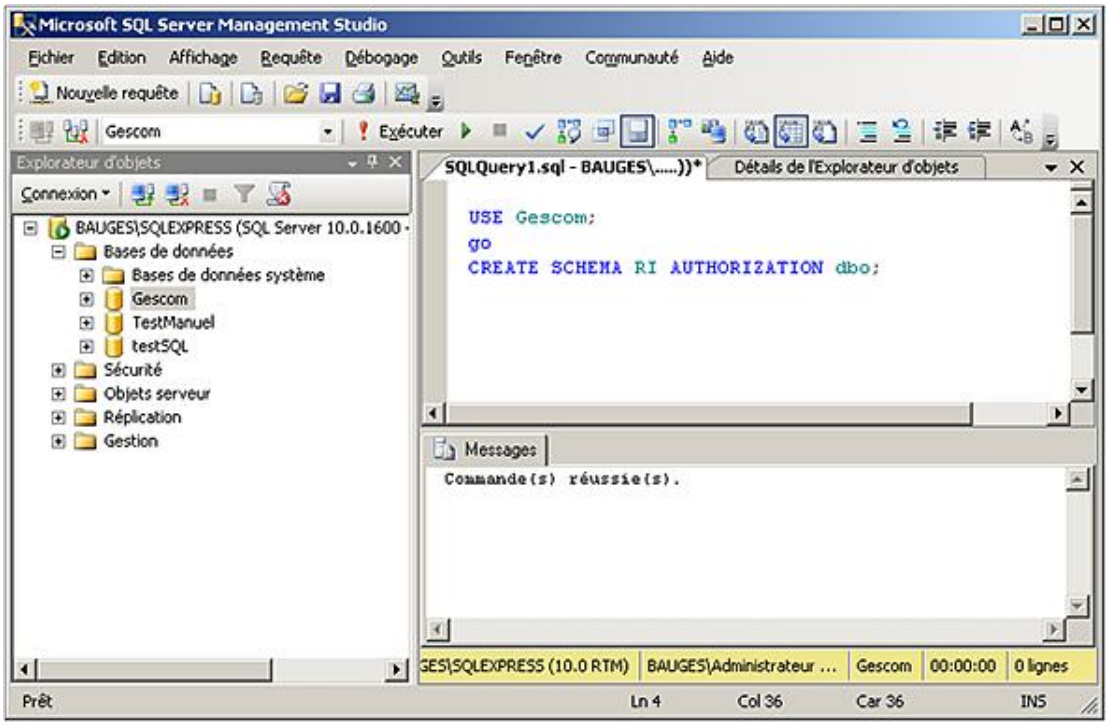

### b. Transact SQL

```
CREATE SCHEMA nomSchema
  AUTHORIZATION nomPropriétaire|
  [defitionDesTables |
   definitionDesVues |
   gestionDePrivilèges]
```

Plus simplement, il est possible de dire qu'un schéma est caractérisé par son nom. L'option AUTHORIZATION permet de définir l'utilisateur de base de données qui est propriétaire du schéma. Un même utilisateur de base de données peut être le propriétaire de plusieurs schémas. Le propriétaire d'un schéma ne doit pas nécessairement avoir comme schéma par défaut un schéma dont il est propriétaire. Il est également possible de créer les tables et les vues d'un schéma dès la construction du schéma.

*Exemple*

*Dans l'exemple suivant, le schéma RI est créé.*




Dans ce second exemple, le schéma *livre* est défini ainsi que des tables et des vues spécifique à ce schéma.

```
CREATE SCHEMA livre AUTHORIZATION dbo
CREATE TABLE articles(reference nvarchar(8) constraint pk_articles
primary key, designation nvarchar(200), prixht money, tva numeric(4,2))
CREATE VIEW catalogue AS select reference, designation, prixht,
tva ttc=prix*(1+tva)/100 FROM articles;
```

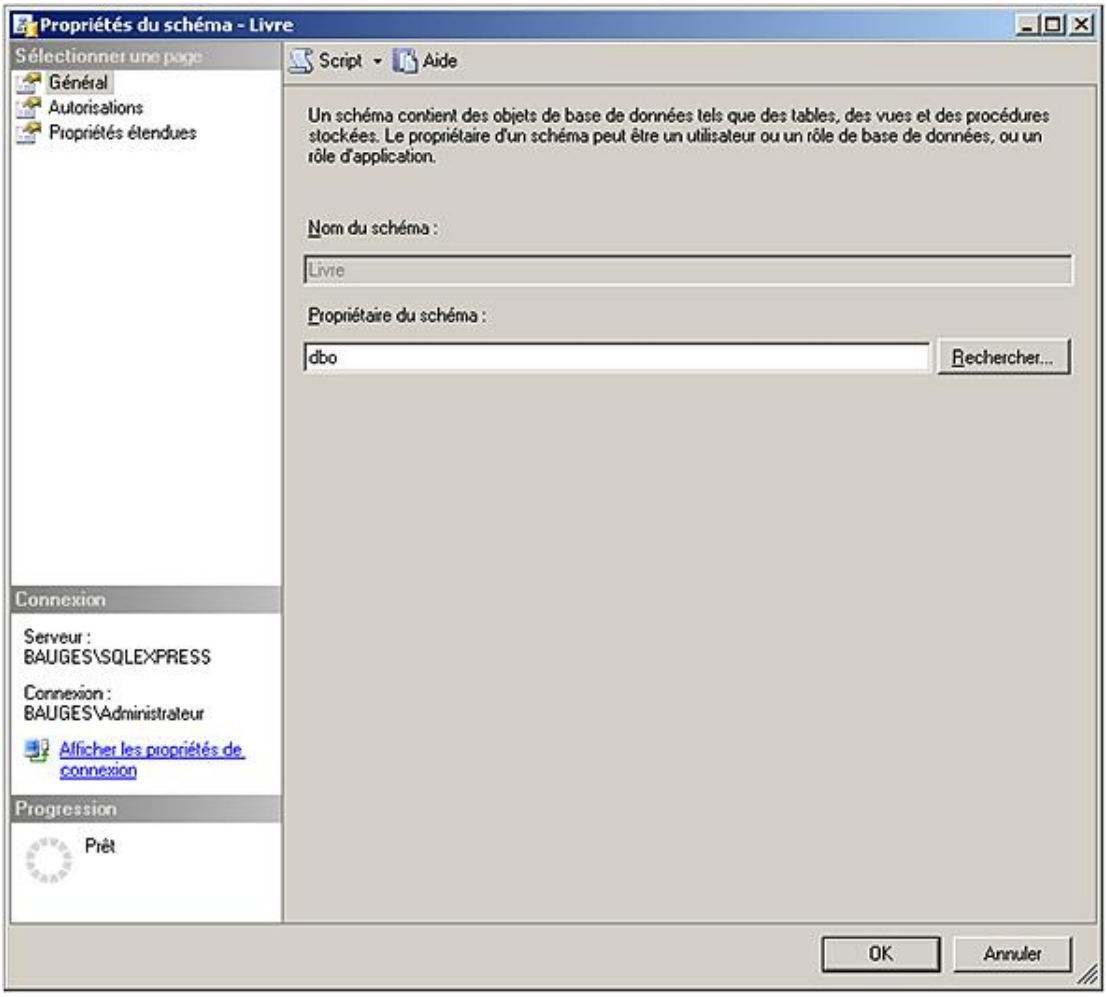
## 2. Modification

La modification d'un schéma consiste à modifier les objets contenus dans ce schéma. Lorsqu'un objet est transféré d'un schéma à un autre, les autorisations relatives à cet objet sont perdues. Le transfert d'un objet entre deux schémas est possible à l'intérieur d'une même base de données.

### a. SQL Server Management Studio

Pour modifier un schéma de base de données, il faut se positionner sur la base de données concernée par la modification et depuis l'explorateur d'objets réaliser la manipulation suivante :

- Développer le nœud **Sécurité** et se positionner sur le nœud **Schémas**.
- Se positionner sur le schéma à modifier.
- Afficher la boîte de propriétés depuis l'option **Propriétés** disponible dans le menu contextuel associé au schéma.




### b. Transact SQL

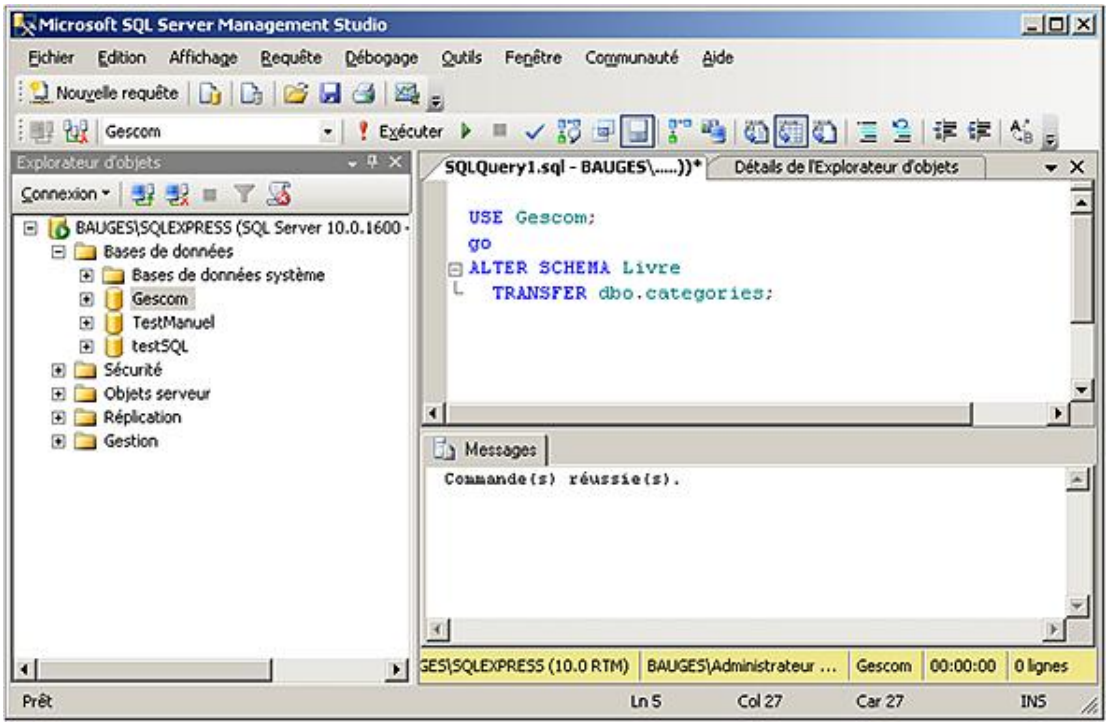
```
ALTER SCHEMA nomSchema TRANSFER nomObjet
```

`nomObjet`

Représente le nom de l'objet qui va être transféré dans le schéma courant. Le nom de cet objet est au format suivant nomSchema.nomCourtObjet.

*Exemple*

*Dans l'exemple suivant, la table catégories présente dans le schéma dbo est transférée vers le schéma Livre.*



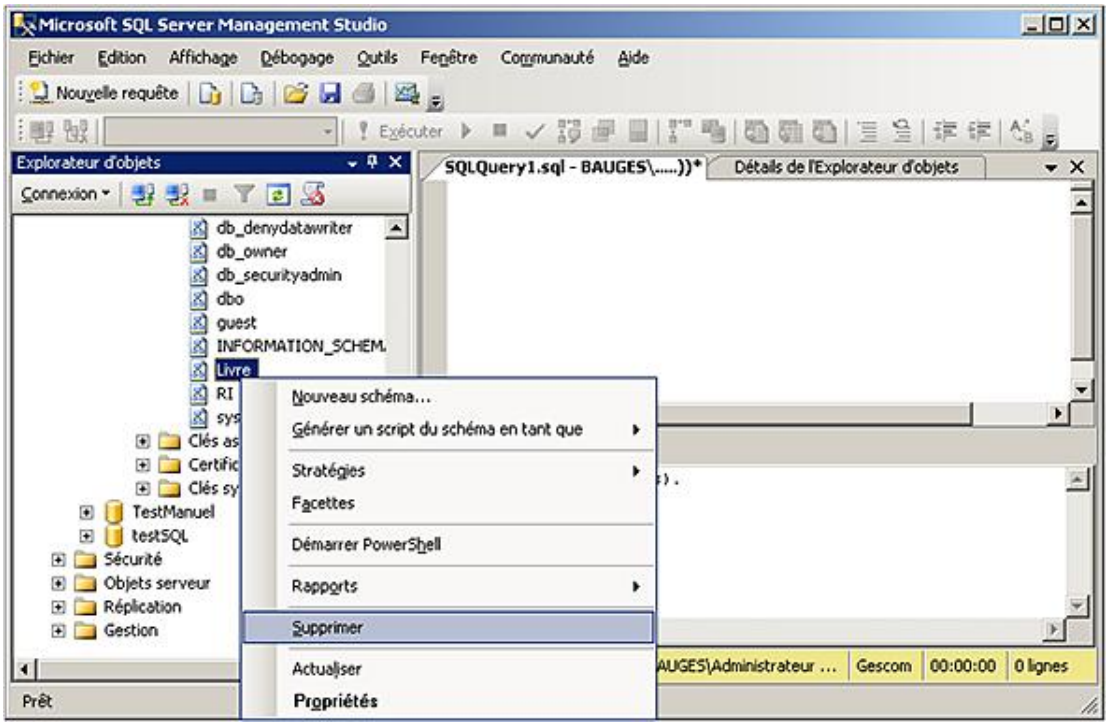

## 3. Suppression

La suppression d'un schéma est possible pour les schémas qui ne contiennent pas d'objet. Par exemple, si le schéma contient des tables ou des vues, il est nécessaire de supprimer ou de transférer vers un autre schéma l'ensemble des tables et des vues avant de pouvoir supprimer le schéma courant.

### a. SQL Server Management Studio

Pour supprimer un schéma de base de données, il faut se positionner sur la base de données concernée par la suppression et depuis l'explorateur d'objets réaliser la manipulation suivante :

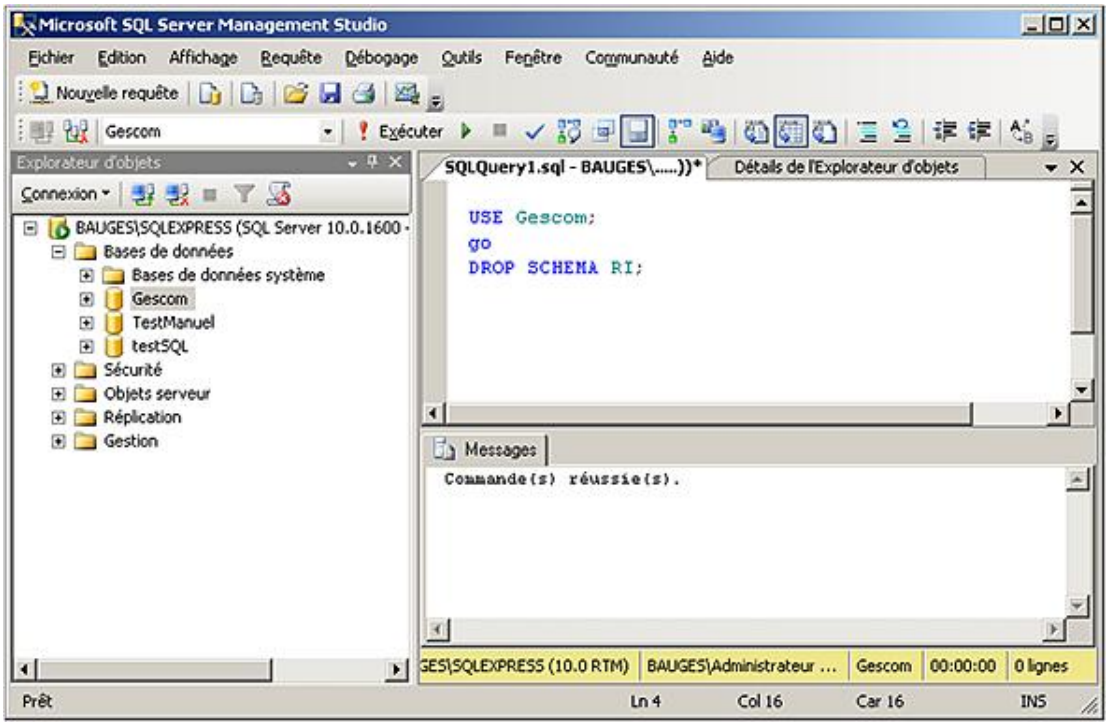
- Développer le nœud **Sécurité** et se positionner sur le nœud **Schéma**.
- Se positionner sur le schéma à supprimer.
- Sélectionner le choix **Supprimer** depuis le menu contextuel associé au schéma.




## b. Transact SQL

```
DROP SCHEMA nomSchema
```

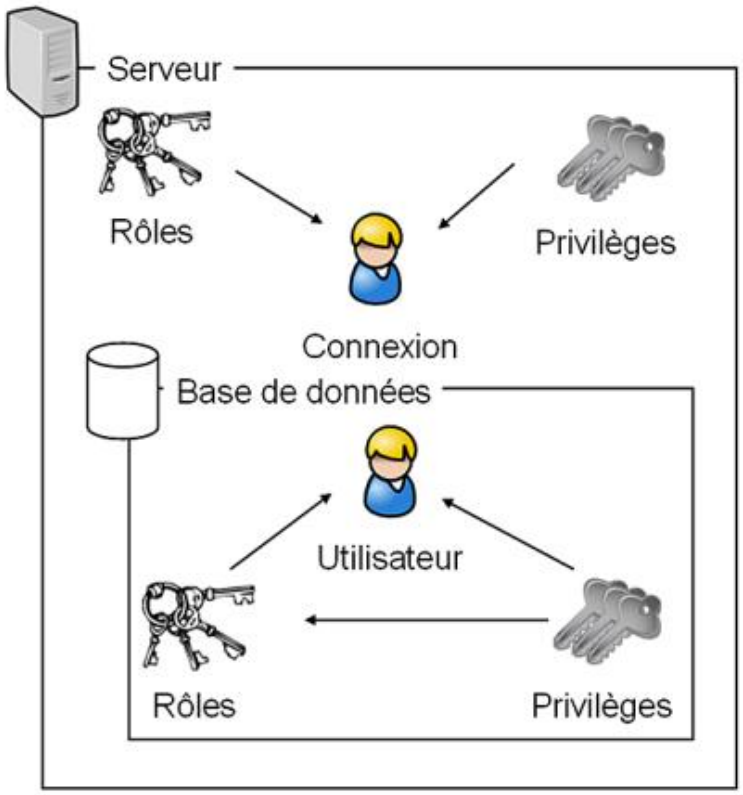
*Exemple*



# Gestion des droits

Tous les utilisateurs de base de données, y compris **guest**, appartiennent au groupe **public**. Les droits qui vont être détaillés ci-dessous peuvent bien sûr être accordés directement à **public**.

Les droits sont organisés de façon hiérarchique par rapport aux éléments sécurisables du serveur.




Il est possible de gérer l'attribution de privilèges au niveau du serveur, de la base de données, du schéma ou bien directement de l'objet. Ainsi, les privilèges peuvent être accordés soit à un utilisateur de base de données, soit à une connexion.

SQL Server gère les privilèges avec trois types de mots clés :

- GRANT
- REVOKE
- DENY

C'est-à-dire qu'un privilège peut être accordé (GRANT) ou bien retiré (REVOKE) s'il a été accordé. L'instruction DENY permet d'interdire l'utilisation d'un privilège particulier même si le privilège en question a été accordé soit directement, soit par l'intermédiaire d'un rôle.

## 1. Droits d'utilisation d'instructions

Les droits d'utilisation des instructions SQL pour créer de nouveaux objets au sein de la base sont des autorisations pour réaliser l'exécution de certains ordres SQL. Un utilisateur qui dispose de tels droits est capable par exemple de créer ses propres tables, ses procédures... L'accord de ces droits peut être dangereux et comme pour tous les droits, doivent être accordés uniquement lorsque cela est nécessaire.

Les droits principaux d'instructions disponibles sont :

- CREATE DATABASE
- CREATE PROCEDURE

- CREATE FUNCTION
- CREATE TABLE
- BACKUP DATABASE
- CREATE VIEW
- BACKUP LOG

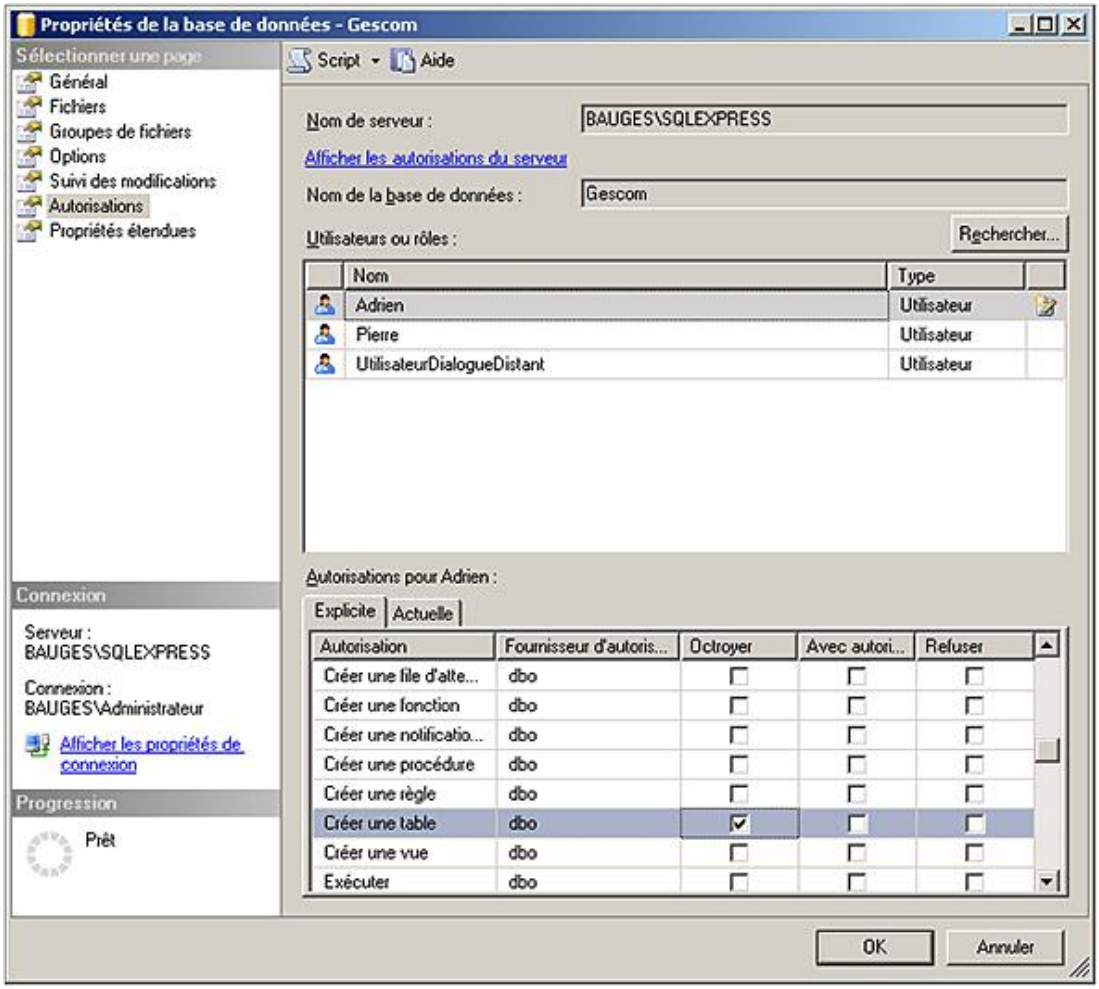
### a. Autoriser

#### SQL Server Management Studio

Ces droits sont administrés au niveau de la base de données par l'intermédiaire de la fenêtre des propriétés.

*Exemple*

*Le privilège CREATE TABLE est accordé à l'utilisateur de base de données Adrien par l'intermédiaire de la boîte de propriétés de la base GESCOM.*




#### Transact SQL

L'accord de privilèges s'effectue en utilisant l'instruction GRANT dont la syntaxe est détaillée ci-dessous.

```
GRANT permission [,...]
```

```
    TO utilisateur[,...]
    [ WITH GRANT OPTION ]
```

`permission`

Nom de la ou des permissions concernées par cette autorisation. Il est également possible d'utiliser le mot clé ALL à la place de citer explicitement la ou les permissions accordées. Toutefois ce terme ALL ne permet pas d'accorder des privilèges d'exécution de toutes les instructions mais simplement sur les instructions pour créer des bases de données, des tables, des procédures, des fonctions, des tables vues ainsi que d'effectuer des sauvegardes de la base et du journal.

`utilisateur`

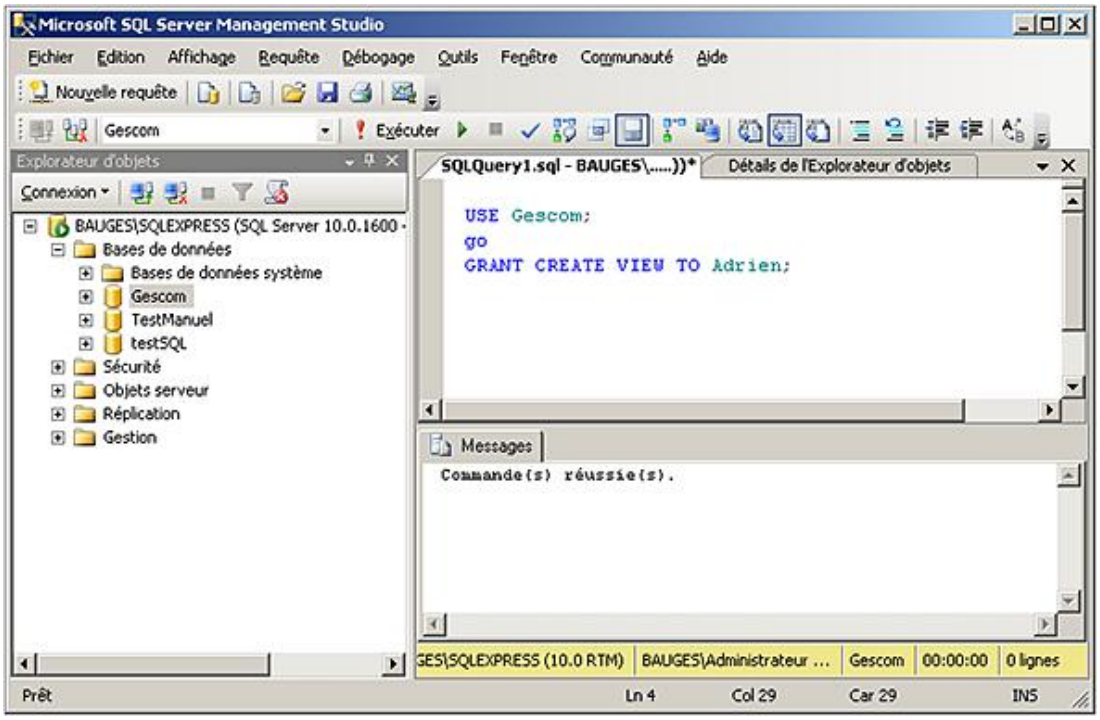
Nom de l'utilisateur ou des utilisateurs de base de données qui reçoivent les permissions.

`WITH GRANT OPTION`

Si la permission est reçue avec ce privilège, alors l'utilisateur peut accorder la permission à d'autres utilisateurs de base de données.

*Exemple*

*Dans l'exemple suivant, le privilège CREATE VIEW est accordé à l'utilisateur de base de données Adrien.*




## b. Retirer

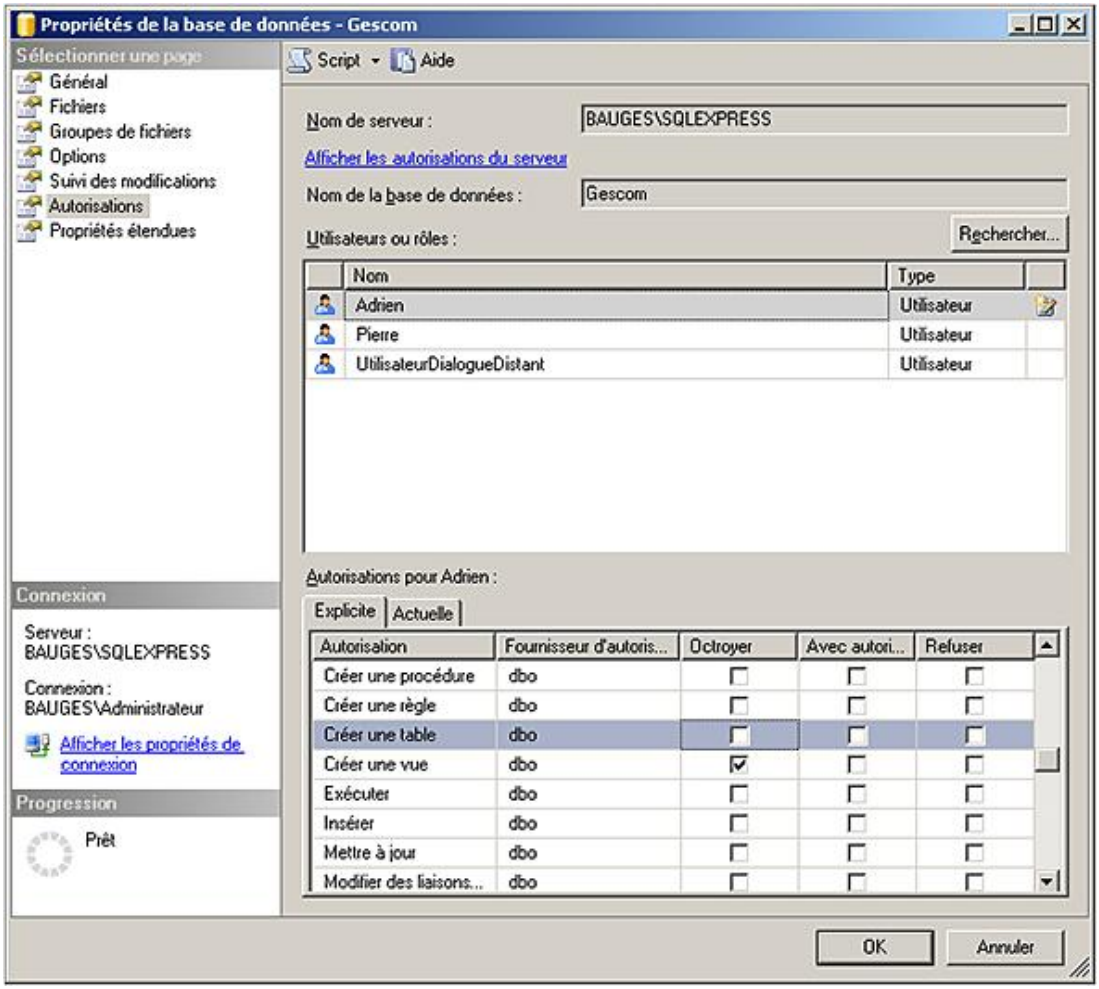
Il est possible de retirer un privilège qui a été accordé à une entité de sécurité. Si le privilège n'a pas été accordé à l'entité de sécurité, l'instruction est sans effet.

### SQL Server Management Studio

La gestion des privilèges s'effectue au niveau des propriétés de la base de données.

*Exemple*

*La permission CREATE TABLE est retirée à l'utilisateur ADRIEN.*




## Transact SQL

```
REVOKE [GRANT OPTION FOR]
       permission [,...]
       FROM utilisateur [,...]
       [CASCADE]
```

`GRANT OPTION FOR`

Si la permission a été accordée avec le privilège d'administration GRANT OPTION, il est possible par l'intermédiaire de cette option de retirer simplement le paramètre d'administration.

`permission`

Liste des permissions à retirer.

`utilisateur`

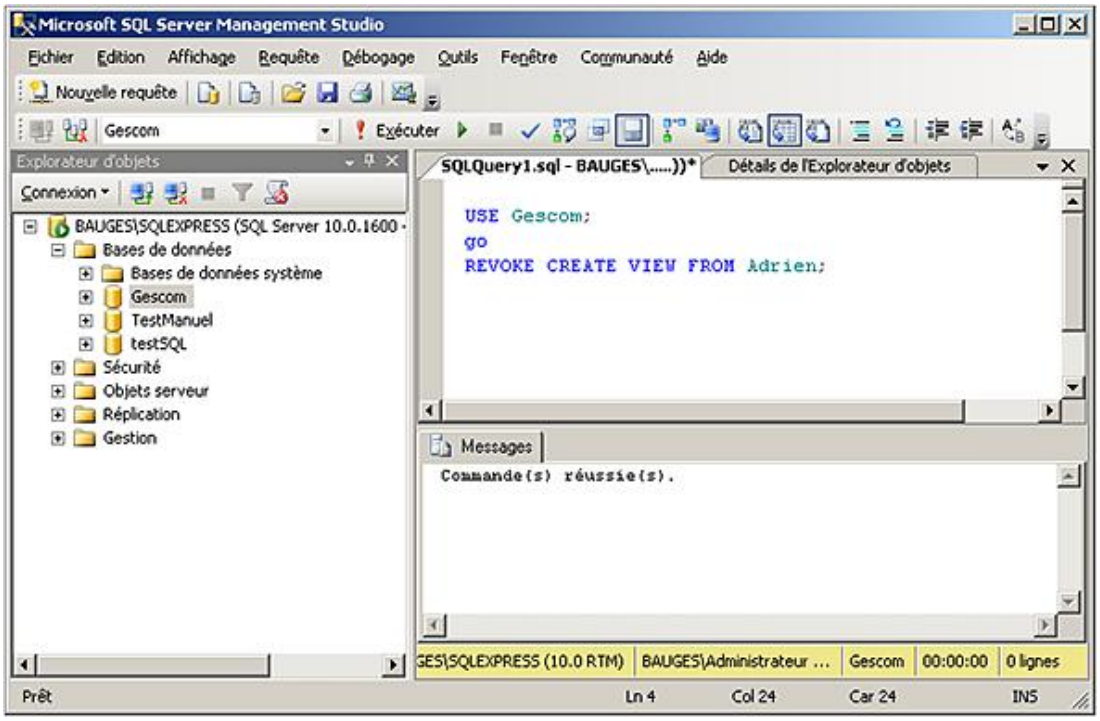
Liste des utilisateurs concernés par cette suppression.

`CASCADE`

Si la permission retirée a été accordée avec le privilège d'administration WITH GRANT OPTION, l'option CASCADE permet de retirer le privilège aux utilisateurs qui l'ont reçu par l'intermédiaire de l'utilisateur qui est concerné par la suppression initiale de privilège.

*Exemple*

*Le privilège CREATE VIEW est retiré à l'utilisateur Adrien.*




### c. Interdire

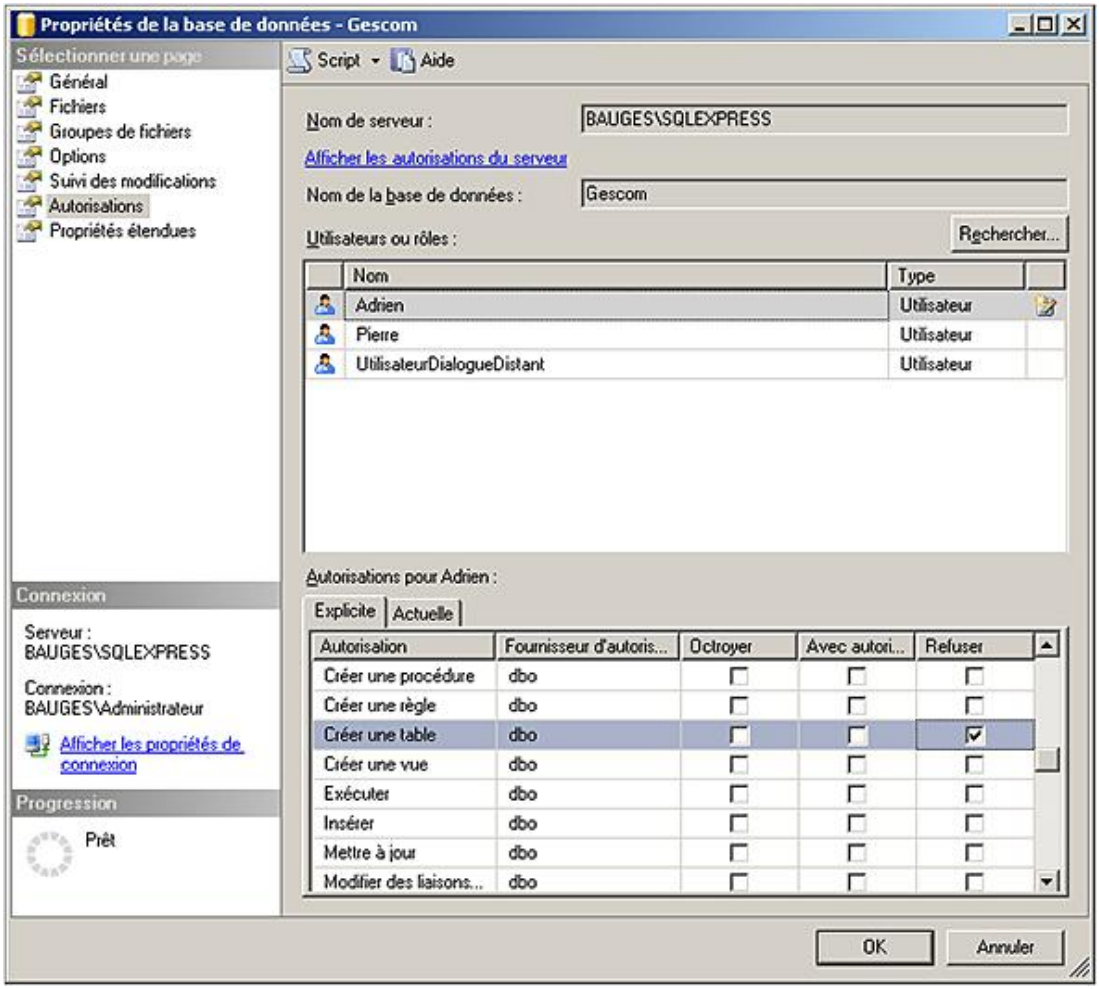
L'instruction DENY permet d'interdire à un utilisateur l'utilisation d'un privilège, même s'il en reçoit la permission soit directement, soit par son appartenance à un groupe.

#### SQL Server Management Studio

Comme pour l'accord et la suppression, ce type de privilège est géré de façon centralisée au niveau des propriétés de la base de données.

*Exemple*

*Par l'intermédiaire des propriétés de la base de données, l'utilisateur Adrien se voit interdire le fait de créer une table.*

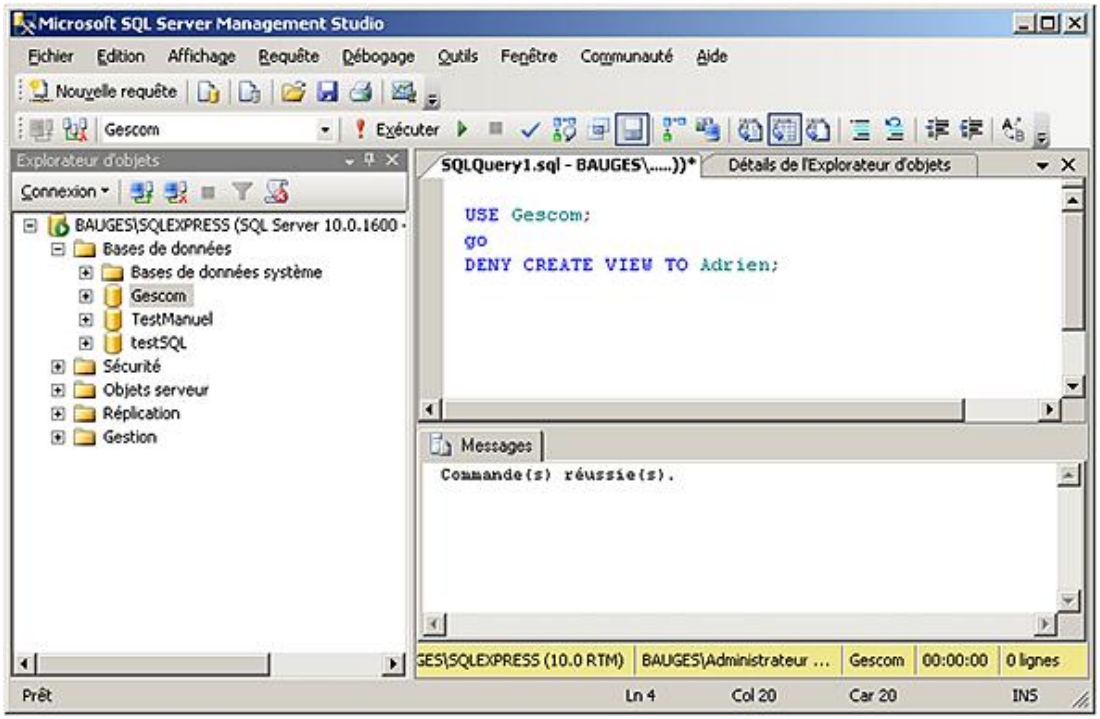



## Transact SQL

```
DENY permission [,...]
      TO utilisateur [,...]
      [CASCADE]
```

*Exemple*

*L'utilisateur Adrien se voit interdire la possibilité de créer des vues.*




## 2. Droits d'utilisation des objets

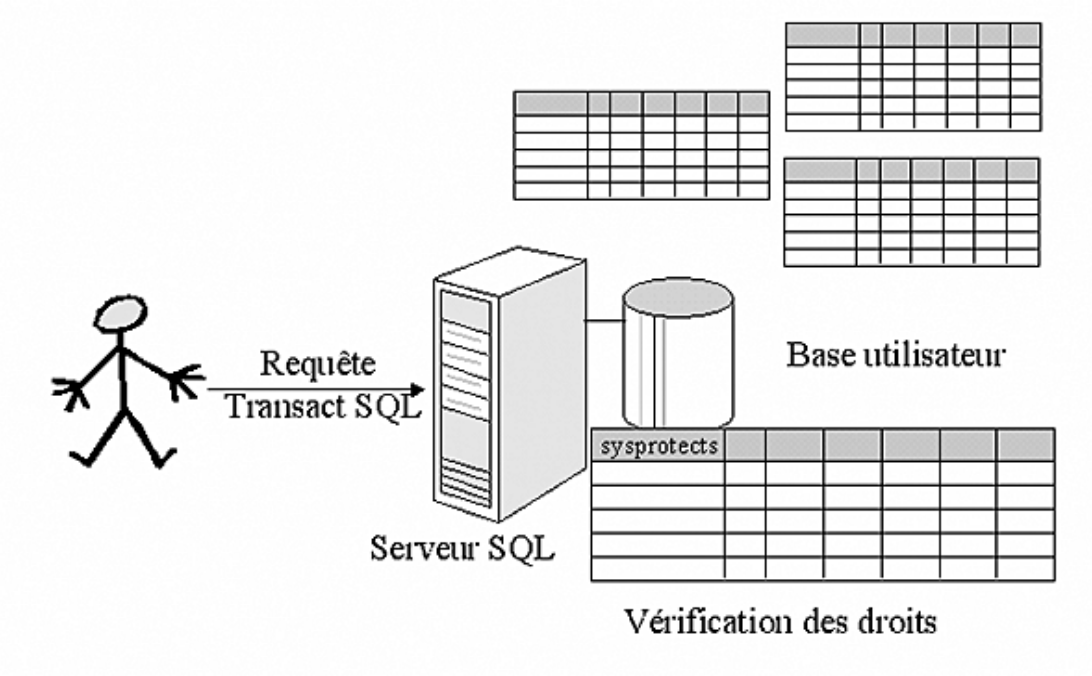
Ces droits permettent de fixer quelles opérations (lecture, modification, ajout ou suppression) l'utilisateur peut réaliser sur des données contenues dans une table, ou bien donner le droit d'exécution d'une procédure stockée. Ces droits sont en général gérés par le propriétaire de l'objet.

En ce qui concerne le droit de lecture des données contenues dans une table (utilisation de l'instruction SELECT), il est possible de préciser quelles colonnes l'utilisateur peut visualiser. Par défaut, il s'agit de toutes les colonnes.

Les principaux droits d'utilisation des objets concernant les tables, vues, procédures stockées correspondent aux ordres :

- INSERT
- UPDATE
- DELETE
- SELECT
- EXECUTE : qui ne s'utilise que pour les procédures stockées.

Les instructions SELECT et UPDATE peuvent être limitées à certaines colonnes de la table ou de la vue. Il est cependant préférable de ne pas trop explorer cette possibilité car il en résulte un plus grand travail administratif au niveau de la gestion des droits. Il est préférable de passer par une vue ou bien une procédure stockée pour limiter l'usage de la table.




*Principe de fonctionnement des autorisations*

### a. Autoriser

#### SQL Server Management Studio

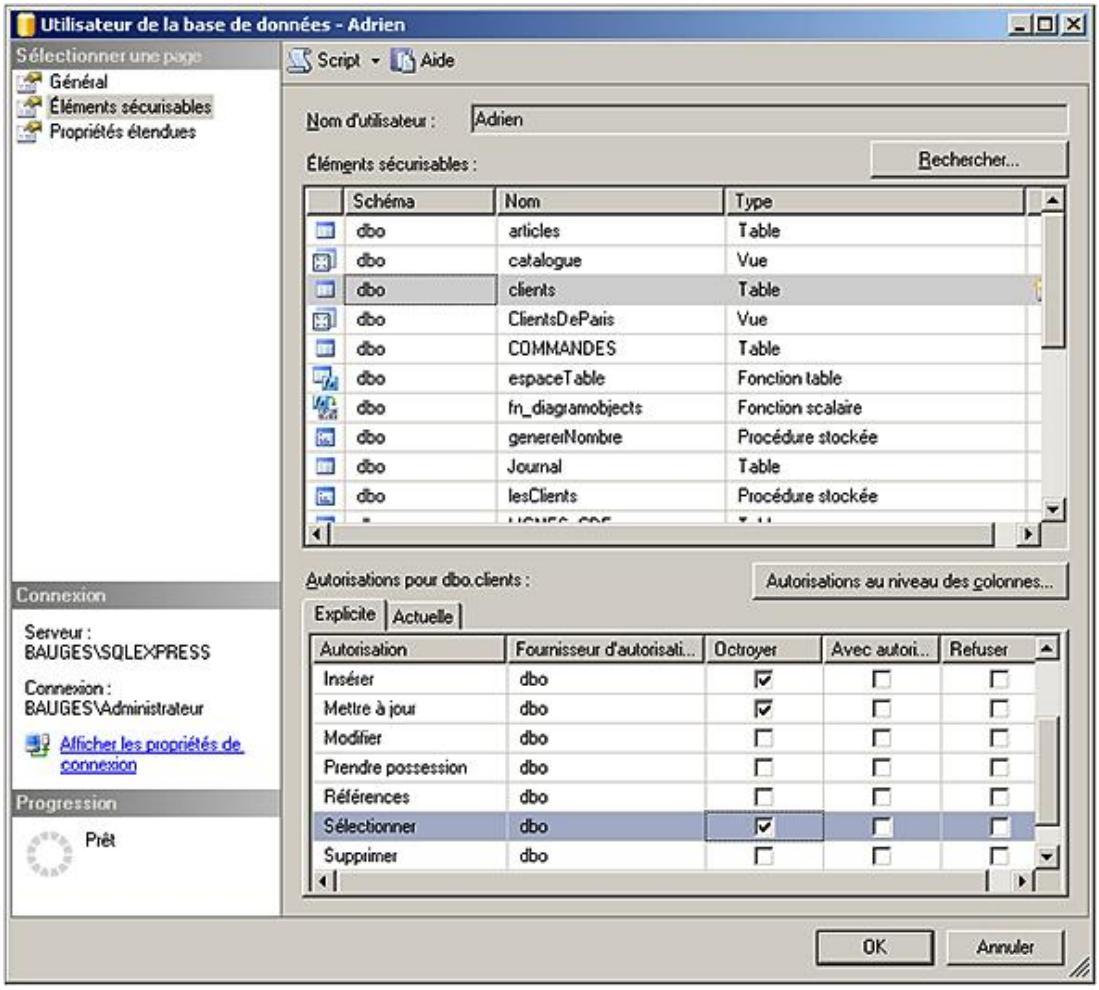
Les privilèges d'utilisation des objets peuvent être gérés à deux niveaux dans SQL Server Management Studio :

- au niveau utilisateur, ce qui permet de connaître les possibilités de travail que possède chaque utilisateur.
- au niveau des objets, afin de connaître quels sont les utilisateurs qui peuvent utiliser l'objet en question et comment ils peuvent l'utiliser.

Dans chacun des deux cas, les autorisations sont gérées par l'intermédiaire de la fenêtre des propriétés.

*Exemple*

*Dans l'exemple suivant, l'utilisateur Adrien se voit accorder la possibilité d'exécuter l'instruction SELECT sur la table des CLIENTS du schéma dbo.*




## Transact SQL

```
GRANT {ALL [PRIVILEGES]|permission[(colonne,[,...])] [,...]}
    ON objet[,...]
    TO utilisateur [,...]
    [WITH GRANT OPTION ]
```

`ALL`

Permet d'accorder tous les privilèges d'utilisation de l'objet. Les privilèges accordés sont toujours fonction du type de l'objet concerné par cet accord de privilège.

`PRIVILEGES`

Le mot clé a été ajouté pour le respect de la norme ANSI-92. Il n'apporte aucune modification quant à l'utilisation du mot clé ALL.

`permission`

Permet de spécifier la ou les opérations qui seront accordées aux utilisateurs.

`objet`

Correspond au nom complet de l'objet ou des objets sur lesquels porte l'accord de privilège d'utilisation.

`utilisateur`

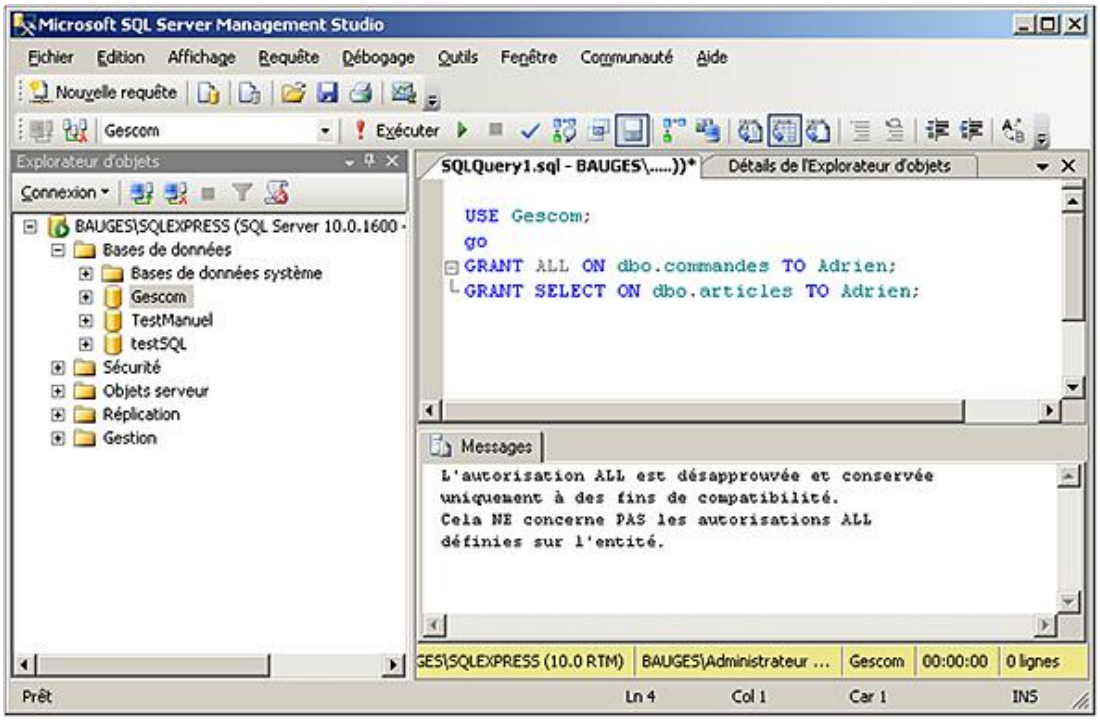
Correspond à l'utilisateur, ou plus exactement à l'entité de sécurité qui va pouvoir bénéficier du ou des privilèges.

`WITH GRANT OPTION`

Le privilège est accordé avec une option d'administration qui autorise le bénéficiaire du privilège à accorder ce même privilège à d'autres utilisateurs de la base de données.

*Exemple*

*Dans l'exemple ci-dessous, l'utilisateur Adrien reçoit tous les privilèges d'utilisation sur la table des commandes ainsi que la possibilité de mener des requêtes d'interrogation sur la table des articles.*




## b. Retirer

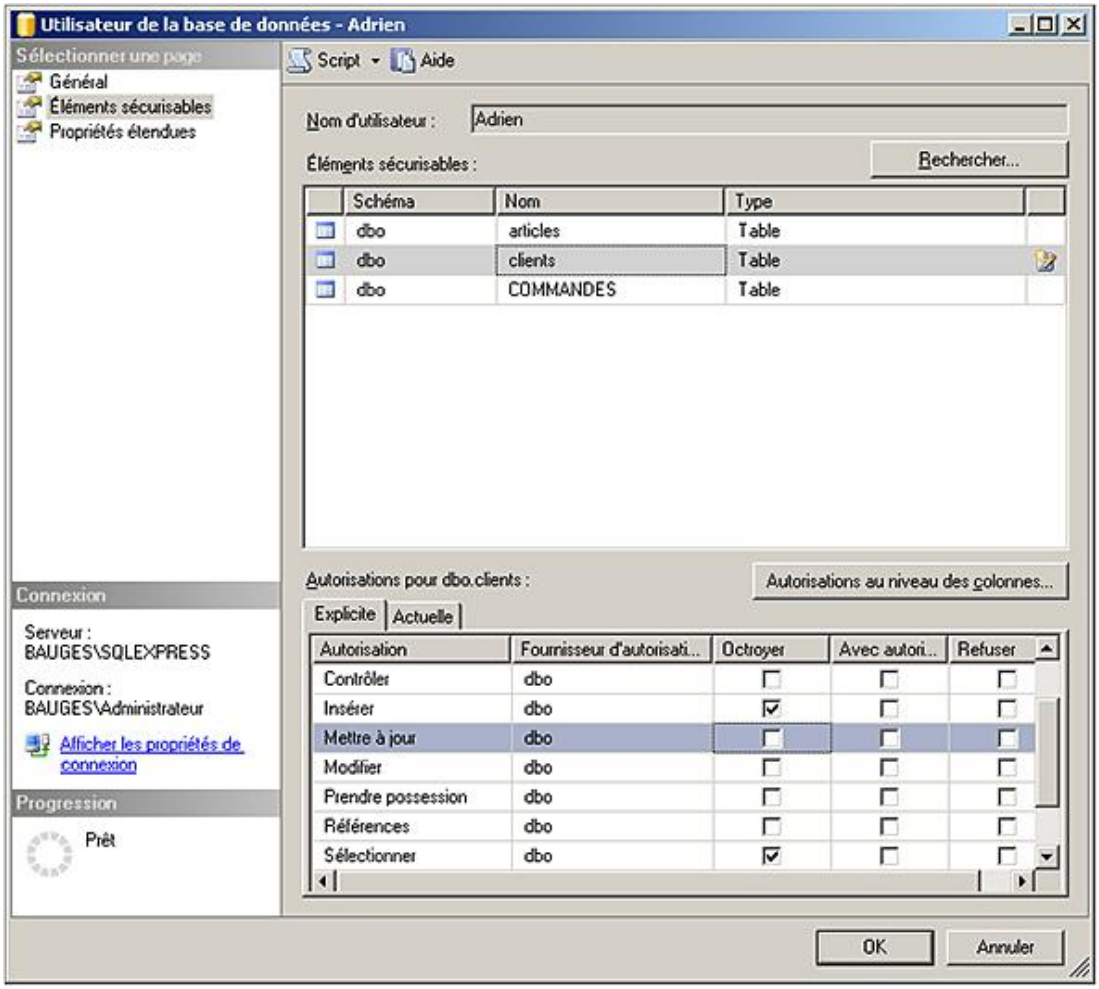
Lorsqu'une permission a été accordée, il est possible de retirer cet accord. Il n'est pas possible de retirer l'utilisation d'une permission à une entité de sécurité si cette permission n'a pas été accordée à cette même entité de sécurité.

### SQL Server Management Studio

C'est par l'intermédiaire de la fenêtre des propriétés de l'entité de sécurité ou bien de l'objet, qu'il est possible de retirer une permission qui a été accordée précédemment.

*Exemple*

*À partir des propriétés de la table dbo.CLIENTS, l'utilisateur ADRIEN se voit retirer la possibilité d'effectuer des requêtes de type UPDATE sur cette table.*




## Transact SQL

```
REVOKE [GRANT OPTION FOR ]
      {ALL [PRIVILEGES]|permission[(colonne,[,...])] [,...]}
       ON object [(colonne [,...])]
      {FROM | TO} utilisateur [,...]
   [ CASCADE ]
```

`GRANT OPTION FOR`

Avec cette option, il est possible de retirer simplement le privilège d'administration du privilège.

`permission`

Permet de préciser la ou les permissions concernées par le retrait. Comme pour l'accord, il est possible de préciser les colonnes concernées par le retrait, mais la gestion d'un tel niveau de droit est lourde.

`objet`

Il faut préciser le nom complet de l'objet au sein de la base de données, c'est-à-dire nomSchema.nomObjet.

`FROM, TO`

Ces deux termes sont synonymes. L'usage habituel consiste à utiliser TO lors de l'accord d'un privilège et FROM lors du retrait d'un privilège.

`utilisateur`

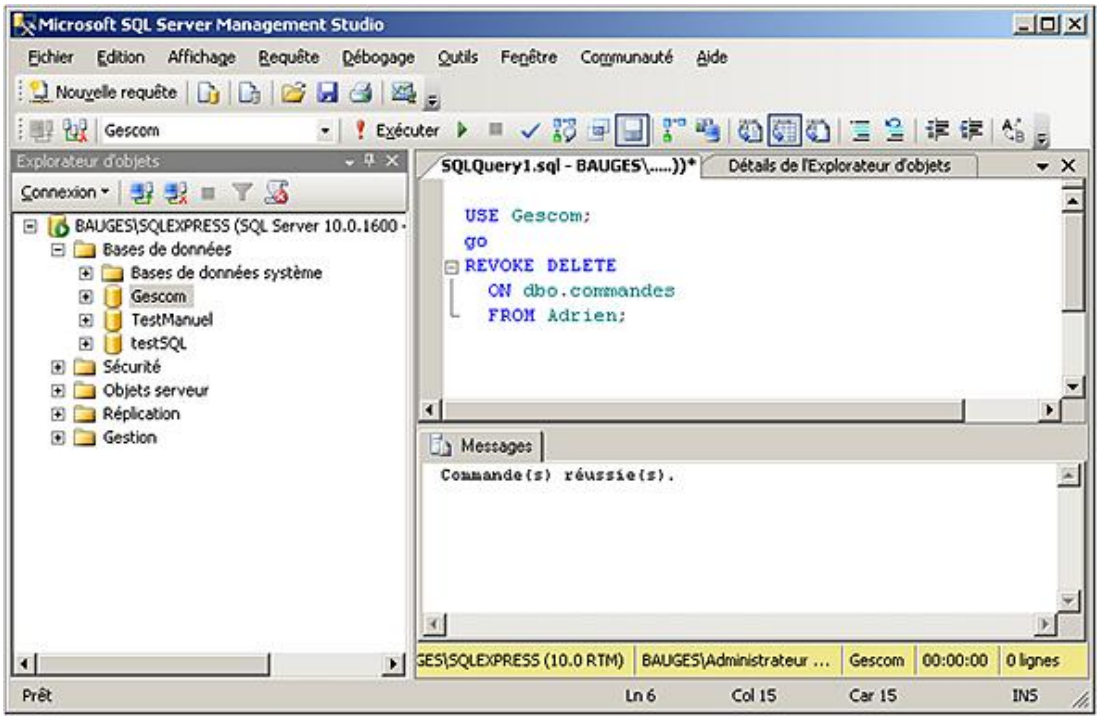
Il s'agit très exactement de l'entité de sécurité à qui le privilège est retiré. Cette entité de sécurité est le plus souvent un utilisateur, mais il peut s'agir d'un rôle par exemple.

CASCADE

Lors du retrait d'une permission accordée avec le privilège d'administration, cette option permet de cascader le retrait, c'est-à-dire d'effectuer le retrait de la permission à toutes les entités de sécurité qui l'ont reçue de la part de celle qui est concernée initialement par le retrait.

*Exemple*

*Dans l'exemple suivant, l'utilisateur Adrien se voit retirer la possibilité de supprimer des COMMANDES.*




## c. Interdire

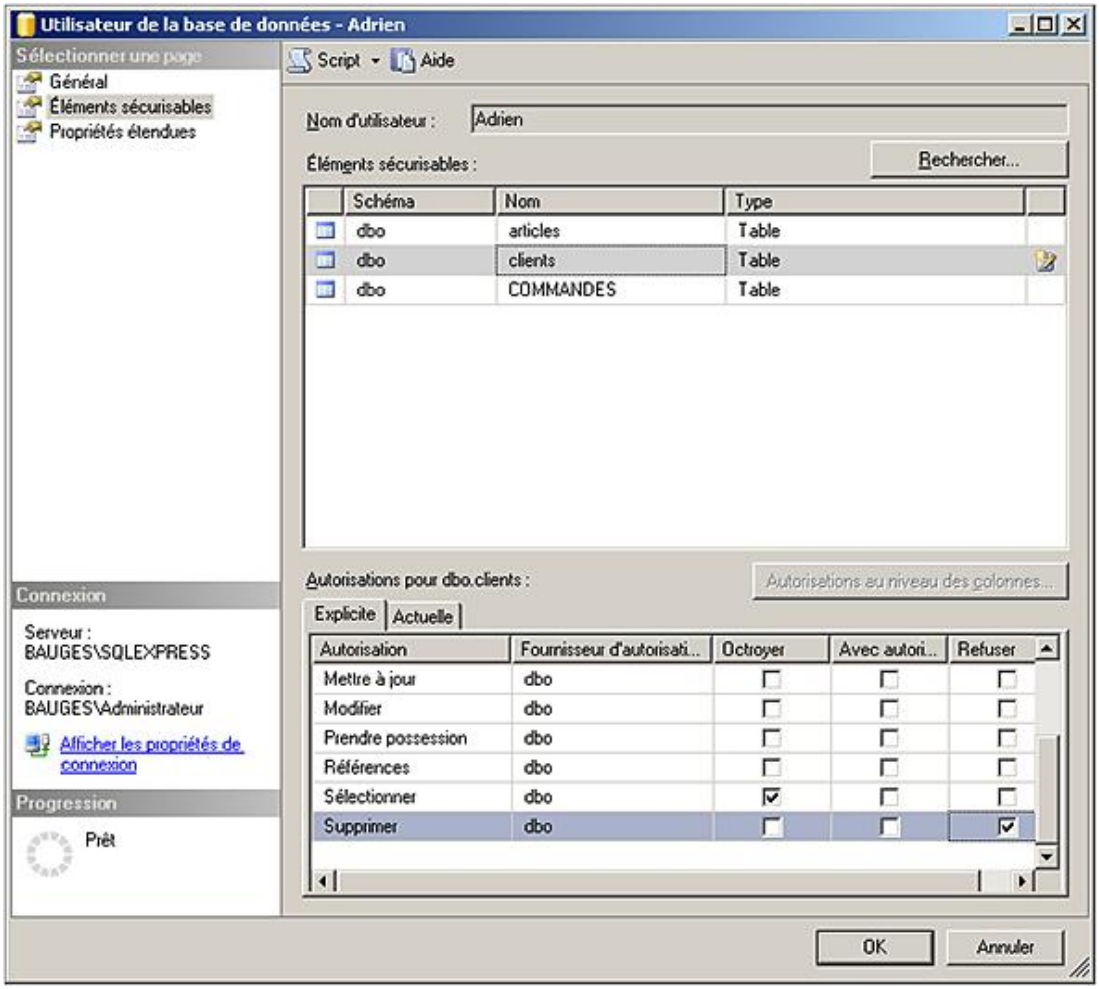
L'interdiction d'utiliser une permission d'utilisation d'un objet est une instruction plus forte que la suppression car elle peut être appliquée à une entité de sécurité, même si cette dernière n'a pas encore reçu de privilège d'utilisation de l'objet de façon directe ou non.

### SQL Server Management Studio

Comme pour l'ajout ou bien le retrait, l'interdiction va être gérée par les fenêtres de propriétés, soit celle de l'entité de sécurité, soit par celle de l'objet. Le choix d'une solution par rapport à l'autre dépend du type de vue que l'on souhaite adopter et du type d'opération que l'on souhaite faire. Est-ce que l'on souhaite, par exemple, mieux contrôler l'utilisation d'une table, ou bien les actions que peut mener un utilisateur de base de données ?

*Exemple*

*Dans l'exemple suivant, l'utilisateur ADRIEN se voit interdire la possibilité de supprimer des informations sur la table dbo.CLIENTS, par l'intermédiaire de la fenêtre de propriétés de la table.*




## Transact SQL

```
DENY {ALL [PRIVILEGES]|permission[(colonne,[,...])] [,...]}
      ON object [(colonne [,...])]
      TO utilisateur [,...]
     [CASCADE]
```

`colonne`

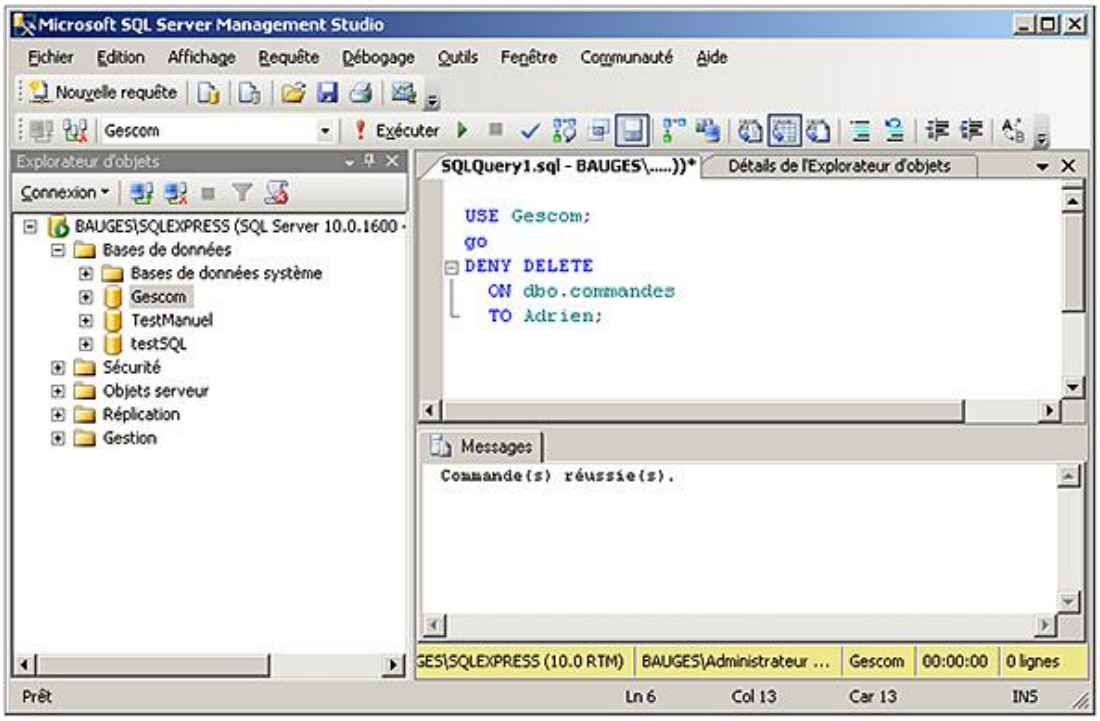
L'interdiction au niveau colonne est sans effet sur une autorisation accordée au niveau de la table. Cette inconsistance de sécurité est maintenue dans SQL Server 2005 pour des raisons de compatibilité antérieure.

`CASCADE`

L'interdiction est transmise aux entités de sécurité qui ont reçu la permission d'utiliser ce privilège par l'intermédiaire de l'entité de sécurité à qui l'utilisation de ce privilège est interdite.

*Exemple*

*Dans l'exemple suivant, l'utilisateur de base de données ADRIEN se voit interdire la possibilité de supprimer des informations depuis la table dbo.COMMANDES.*




## 3. Droits au niveau base de données

Les privilèges accordés au niveau base de données donnent la possibilité à l'utilisateur qui reçoit ce privilège de réaliser des actions qui ont une portée sur l'ensemble de la base de données. Par exemple, le privilège ALTER ANY USER permet à un utilisateur de créer, supprimer et modifier n'importe quel compte de la base de données.

Au niveau base de données, il est possible d'accorder des privilèges à un utilisateur mais également à un schéma, un assembly et à un objet service broker.

La gestion de ces privilèges utilise soit les instructions Transact SQL GRANT, REVOKE et DENY, soit peut être effectuée de façon graphique à partir des propriétés de la base de données.

---

Seules les méthodes pour accorder des privilèges sont illustrées ici, les opérations de suppression et d'interdiction sont très semblables à celles illustrées lors des deux points précédents.

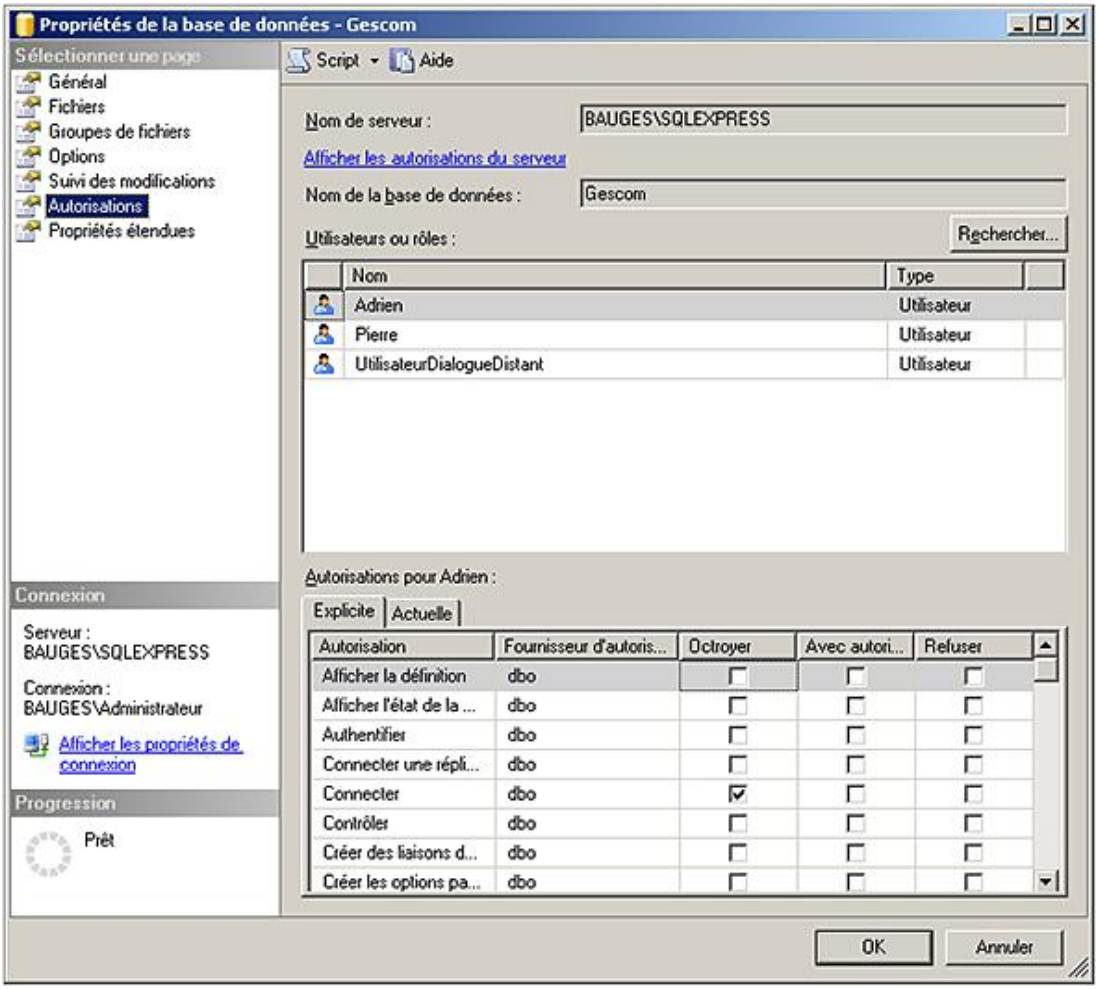
---

### SQL Server Management Studio

Les droits relatifs à l'utilisation peuvent être attribués de la façon suivante depuis SQL Server Management Studio :

- Depuis l'explorateur d'objets, développer le nœud représentant la base de données concernée par la modification de privilège.

- Afficher les propriétés de la base en effectuant le choix **Propriétés** depuis le menu contextuel.




Il est également possible d'accorder ces permissions de façon graphique en allant modifier les propriétés des utilisateurs de base de données.

## Transact SQL

La même opération peut être réalisée en Transact SQL à l'aide de l'instruction GRANT.

```
GRANT permissionBaseDeDonnées [ ,... ]
TO utilisateur [ ,...n ]
[ WITH GRANT OPTION ]
[ AS { group | role } ]
```

`permissionBaseDeDonnées`

La ou les permissions de base de données accordées.

`Utilisateur`

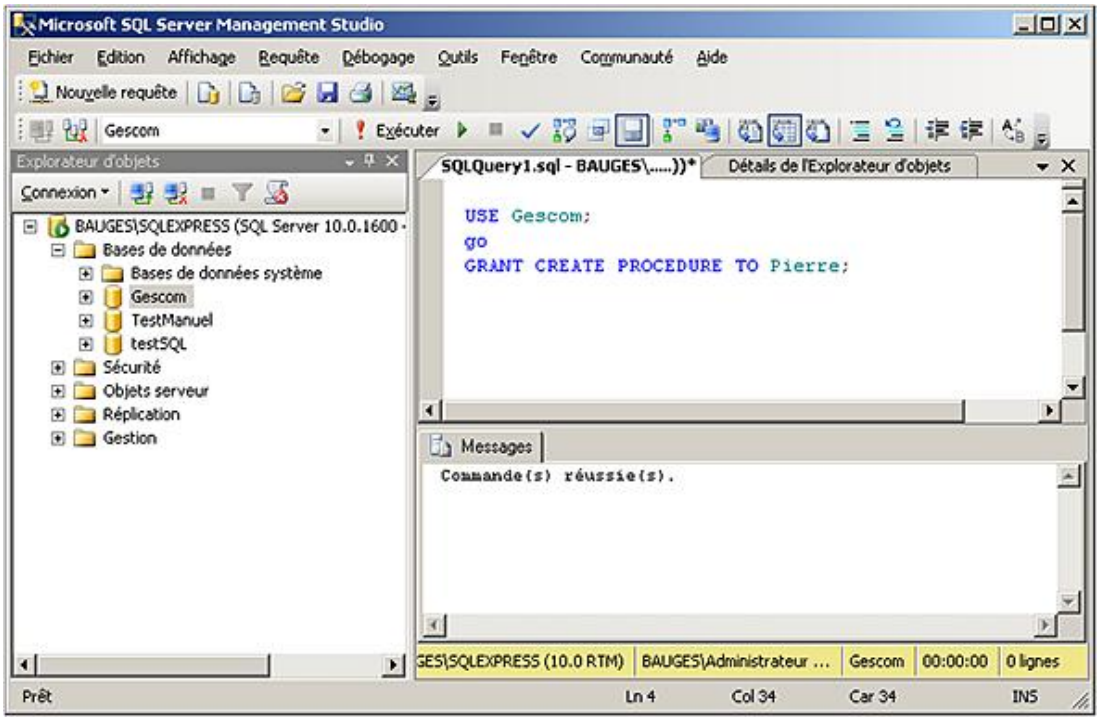
Compte d'utilisateur de base de données qui reçoit le privilège.

`AS {group|role}`

Contexte de sécurité utilisé pour pouvoir accorder le privilège.

*Exemple*

*Dans l'exemple suivant, le privilège CREATE PROCEDURE est accordé à l'utilisateur Pierre.*




Les privilèges qu'il est possible d'accorder à ce niveau sont :

| | |
|---|---|
| ALTER | CREATE DATABASE |
| ALTER ANY APPLICATION ROLE | CREATE DATABASE DDL EVENT NOTIFICATION |
| ALTER ANY ASSEMBLY | CREATE DEFAULT |
| ALTER ANY ASYMMETRIC KEY | CREATE FULLTEXT CATALOG |
| ALTER ANY CERTIFICATE | CREATE FUNCTION |
| ALTER ANY CONTRACT | CREATE MESSAGE TYPE |
| ALTER ANY DATABASE DDL TRIGGER | CREATE PROCEDURE |
| ALTER ANY DATABASE EVENT NOTIFICATION | CREATE QUEUE |
| ALTER ANY DATASPACE | CREATE REMOTE SERVICE BINDING |
| ALTER ANY FULLTEXT CATALOG | CREATE ROLE |
| ALTER ANY MESSAGE TYPE | CREATE ROUTE |
| ALTER ANY REMOTE SERVICE BINDING | CREATE RULE |
| ALTER ANY ROLE | CREATE SCHEMA |
| ALTER ANY ROUTE | CREATE SERVICE |
| ALTER ANY SCHEMA | CREATE SYMMETRIC KEY |
| ALTER ANY SERVICE | CREATE SYNONYM |
| ALTER ANY SYMMETRIC KEY | CREATE TABLE |
| ALTER ANY USER | CREATE TYPE |

| | |
|---|---|
| AUTHENTICATE | CREATE VIEW |
| BACKUP DATABASE | CREATE XML SCHEMA COLLECTION |
| BACKUP LOG | DELETE |
| CHECKPOINT | EXECUTE |
| CONNECT | INSERT |
| CONNECT REPLICATION | REFERENCES |
| CONTROL | SELECT |
| CREATE AGGREGATE | SHOWPLAN |
| CREATE ASSEMBLY | SUBSCRIBE QUERY NOTIFICATIONS |
| CREATE ASYMMETRIC KEY | TAKE OWNERSHIP |
| CREATE CERTIFICATE | UPDATE |
| CREATE CONTRACT | VIEW DATABASE STATE |
| | VIEW DEFINITION |

## 4. Droits au niveau du serveur

SQL Server permet d'attribuer des privilèges au niveau du serveur. Ces privilèges ne sont pas accordés à un utilisateur de base de données mais à une connexion.

Comme pour les droits d'utilisation des objets et des instructions, il est possible d'accorder ces privilèges avec une option d'administration. Ainsi, la connexion qui possède ce droit peut accorder ce même droit à une ou plusieurs autres connexions.

Les différents droits qu'il est possible d'accorder au niveau serveur sont :

| | |
|---|---|
| ADMINISTER BULK OPERATIONS | CONNECT SQL |
| ALTER ANY CONNECTION | CONTROL SERVER |
| ALTER ANY CREDENTIAL | CREATE ANY DATABASE |
| ALTER ANY DATABASE | CREATE DDL EVENT NOTIFICATION |
| ALTER ANY ENDPOINT | CREATE ENDPOINT |
| ALTER ANY EVENT NOTIFICATION | CREATE TRACE EVENT NOTIFICATION |
| ALTER ANY LINKED SERVER | EXTERNAL ACCESS ASSEMBLY |
| ALTER ANY LOGIN | SHUTDOWN |
| ALTER RESOURCES | UNSAFE ASSEMBLY |
| ALTER SERVER STATE | VIEW ANY DATABASE |
| ALTER SETTINGS | VIEW ANY DEFINITION |

| | |
|---|---|
| ALTER TRACE | VIEW SERVER STATE |
| AUTHENTICATE SERVER | |

Toutes les informations relatives aux privilèges accordés au niveau du serveur sont visibles au travers de la vue **sys.server_permissions**.

Il est également possible d'attribuer des droits d'utilisation d'objets définis au niveau du serveur telles que les terminaisons http ou http endpoint.

Pour pouvoir accorder de tels privilèges, il faut travailler sur la base Master.

## 5. Interroger les vues systèmes

Il est possible, en interrogeant les vues système du dictionnaire et plus exactement celles traitant de la sécurité, de connaître les différentes autorisations et interdictions relatives aux différentes entités de sécurité.

Les principales vues sont :

- sys.database_permissions qui permet de lister les différentes autorisations d'utilisation des privilèges.
- sys.database_principals qui permet de lister les différents comptes de sécurité.

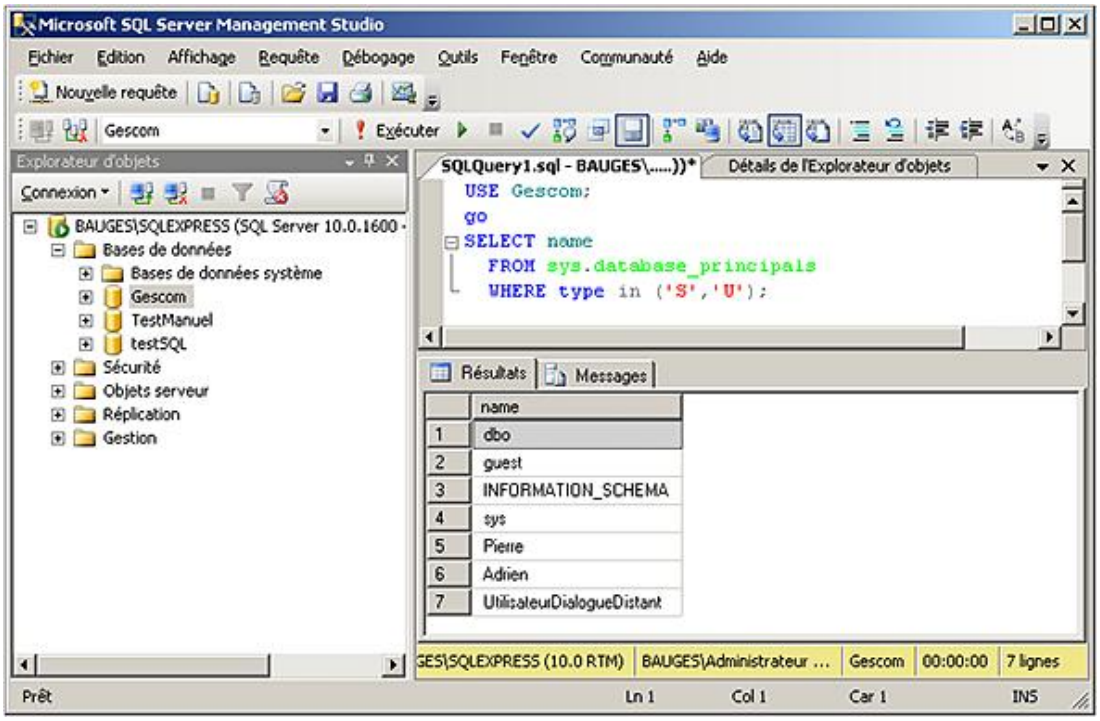
**sys.database_principals**

Cette vue recense toutes les entités de sécurité définies localement à la base de données. Les principales colonnes de cette vue sont :

- **name** nom de l'entité de sécurité. Ce nom est unique à l'intérieur de la base de données.
- **principal_id** identifiant numérique qui permet d'identifier de façon unique une entité de sécurité dans la base de données.
- **type** code relatif au type du contexte de sécurité : utilisateur SQL, utilisateur Windows, rôle, groupe Windows...
- **type_desc** description en toute lettre pour comprendre le type mentionné à la colonne précédente.
- **default_schema_name** nom du schéma associé par défaut à l'entité de sécurité.
- **create_date** date de création de l'entité de sécurité.
- **modify_date** date de la dernière modification sur l'entité de sécurité.
- **owning_principal_id** identifiant du contexte de sécurité qui possède le compte en question. Tous, à l'exception des rôles de base de données, doivent être possédés par **dbo**.
- **sid** ou Security Identifier. Ce champ est renseigné si l'entité de sécurité est liée à un external.
- **is_fixed_role** cet attribut est à 1 lorsque l'entité de sécurité est liée à un rôle fixe prédéfini sur le serveur.

*Exemple*

*La requête suivante permet de connaître les comptes d'utilisateurs définis dans la base GESCOM.*




## sys.database_permissions

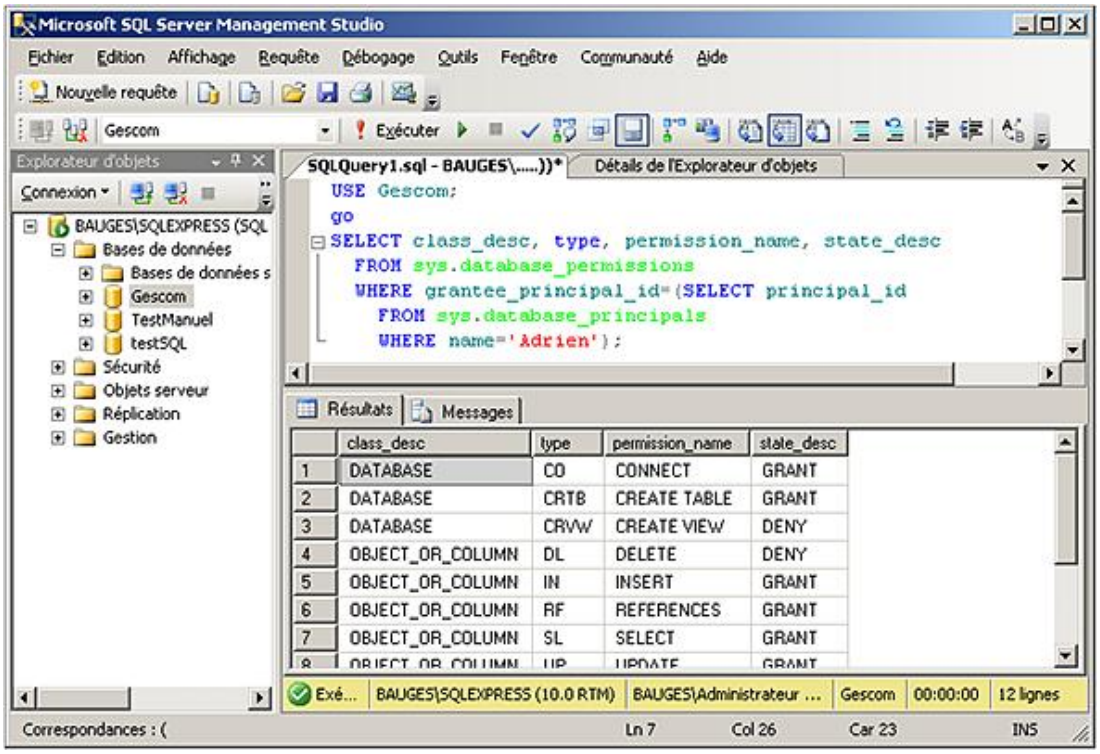
Cette vue permet d'obtenir l'ensemble des informations relatives aux différentes permissions accordées dans la base. Il existe une ligne d'information pour chaque permission accordée à chaque entité de sécurité. Dans le cas de la gestion des permissions au niveau de la colonne, il existe une ligne d'information pour chaque colonne concernée par la permission.

Les principales colonnes de cette vue sont :

- **class** code relatif à la classification du privilège. Est-ce un privilège de niveau base, qui porte sur l'utilisation d'un objet ou d'une colonne...
- **class_desc** description de la classification du privilège.
- **major_id** identifiant de l'objet sur lequel porte la permission.
- **minor_id** identifiant de la colonne sur laquelle porte la permission.
- **grantee_principal_id** identifiant du contexte de sécurité concerné par la permission.
- **grantor_principal_id** identifiant du contexte de sécurité à l'origine de la permission.
- **type** type du privilège.
- **permission_name** nom du privilège concerné par la permission.
- **state** code relatif à l'état de la permission.
- **state_desc** description de l'état actuel de la permission : GRANT, REVOKE, DENY ou bien GRANT_WITH_GRANT_OPTION.

*Exemple*

*À l'aide de la requête suivante, il est facile de connaître les permissions que possède l'utilisateur de base de données ADRIEN.*



# Contexte d'exécution

Le contexte d'exécution correspond à l'ensemble des autorisations actives lors de l'exécution d'une requête, d'un script Transact SQL, d'une procédure, d'une fonction ou d'un trigger. Par défaut, ce sont les privilèges accordés au contexte de sécurité qui exécutent la requête, le script, la procédure, la fonction ou le trigger qui sont utilisés pour exécuter les instructions sous-jacentes. Ce mode de fonctionnement par défaut est, la plupart du temps, le plus adapté et il ne convient pas de le changer. Toutefois, dans certains cas bien précis, il est nécessaire d'assurer la bonne exécution quels que soient les privilèges accordés à l'utilisateur qui est à l'origine de l'exécution. Ce cas se produit par exemple, lorsque personne ne reçoit le privilège d'ajouter (insert) des données dans une table et qu'au contraire une procédure permet de réaliser ce travail. Dans ce cas, la clause EXECUTE AS permet de préciser le contexte d'exécution à utiliser.

Le contexte d'exécution ainsi défini peut correspondre à celui d'un utilisateur nommé expressément, à celui de l'appelant (choix par défaut) ou bien à celui du propriétaire.

*Syntaxe*

```
CREATE PROCEDURE nomProcedure
WITH EXECUTE AS {CALLER|SELF|OWNER|'nomUtilisateur'}
AS corpsDeLaProcedure
```

`CALLER`

Le contexte d'exécution est celui de l'utilisateur qui appelle la procédure ou bien la fonction.

`SELF`

Le contexte d'exécution est celui du créateur de la procédure ou la fonction.

`OWNER`

Le contexte d'exécution de la procédure est celui du propriétaire de la procédure ou de la fonction.

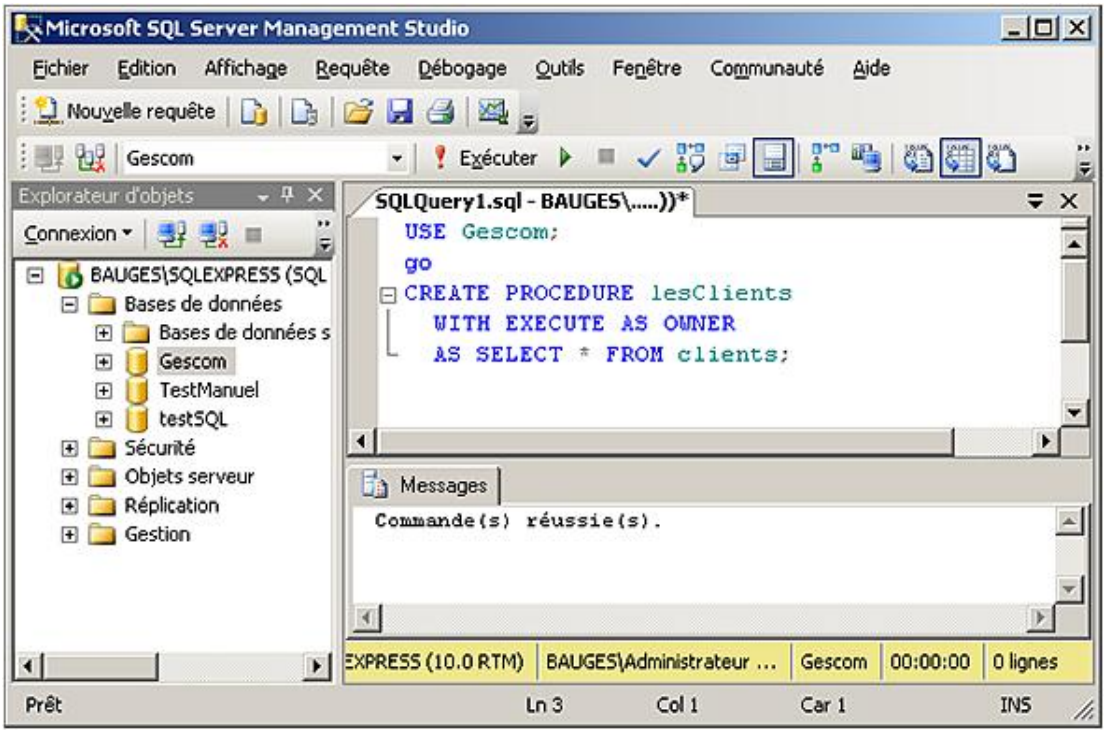
`'nomUtilisateur'`

Le contexte d'exécution de la procédure est celui de l'utilisateur de base de données nommé.

---

Il est également possible de définir le contexte d'exécution sur les déclencheurs de base de données (DML et DDL) ainsi que sur les queues.

---




Dans le cas d'un script, l'instruction EXECUTE AS permet de modifier le contexte d'exécution courant.

# Les rôles

Les rôles sont des ensembles de permissions. Ces ensembles existent à trois niveaux distincts : serveur, base de données et application. Les rôles permettent de regrouper les droits et de gérer plus facilement les différents utilisateurs et les connexions. Il est en effet toujours préférable d'attribuer des droits à des rôles puis d'accorder les rôles aux utilisateurs. Avec une telle structure, l'ajout et la modification de droits ou d'utilisateur sont beaucoup plus simples.

Il est possible de définir un rôle comme un ensemble nommé de privilèges. Pour faciliter la gestion des droits, SQL Server propose des rôles prédéfinis aussi appelés fixes car il n'est pas possible d'ajouter ou bien de supprimer des privilèges dans ces rôles. Ces rôles fixes sont définis à deux niveaux :

- serveur
- base de données

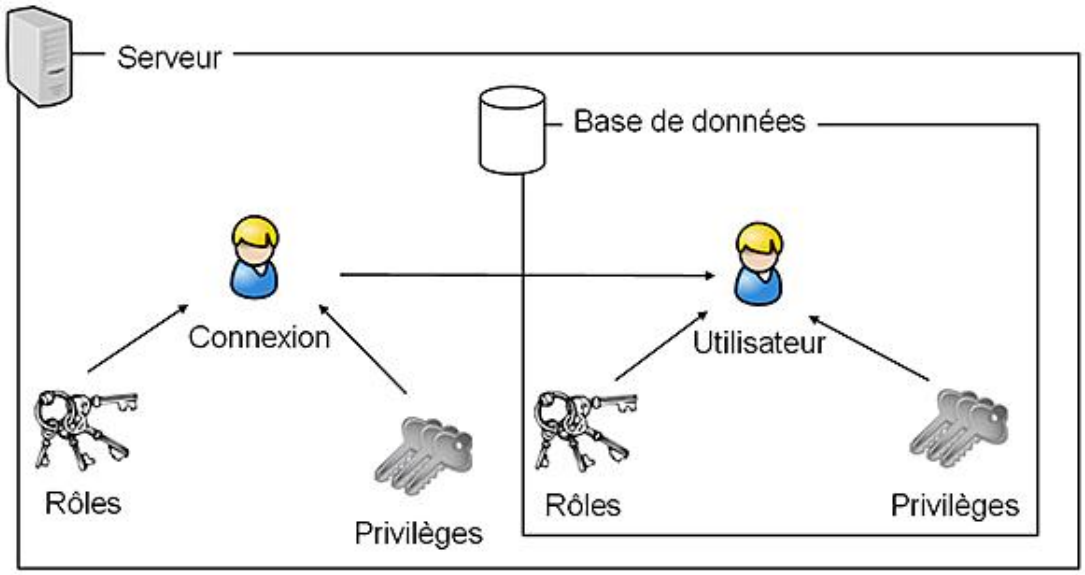
En plus de ces rôles fixes, il est possible de gérer des rôles. Il convient alors de définir un nom unique pour définir un rôle, puis d'accorder un ou plusieurs privilèges en respectant une procédure en tout point similaire à celle utilisée pour accorder des permissions à des utilisateurs. Ces rôles peuvent être définis à deux niveaux :

- base de données
- application

Les rôles permettent une gestion simplifiée des privilèges puisqu'il est possible ainsi de définir des profils types de privilèges, puis d'accorder à chaque utilisateur de base de données, un ou plusieurs profils de façon à lui fournir toutes les autorisations dont il a besoin pour travailler sur la base.

L'utilisateur dispose au final de l'ensemble des privilèges qui sont accordés :

- directement à la connexion utilisée ;
- indirectement par l'intermédiaire d'un rôle fixe de serveur accordé à la connexion ;
- indirectement par l'intermédiaire des rôles accordés à l'utilisateur de base de données ;
- directement à l'utilisateur de base de données.




En complément de ces différents rôles, il existe le rôle **public**. Ce rôle est à part car tous les utilisateurs reçoivent le rôle public et ils ne peuvent pas s'y soustraire. Ce rôle est également particulier car il est possible de lui accorder, retirer ou bien interdire des permissions. Toutes modifications au niveau des permissions faites sur le rôle public sont valables pour tous les utilisateurs. Il n'est donc pas recommandé de travailler avec ce rôle mais, dans certains cas, il

apporte une souplesse de gestion des permissions qui est intéressante.

## 1. Rôles de serveur

Ce sont des rôles prédéfinis qui donnent aux connexions un certain nombre de fonctionnalités. Il n'est pas possible de définir des rôles de serveur, par contre leur utilisation facilite la gestion en cas de nombreux utilisateurs.

Les rôles de serveur sont au nombre de huit :

**sysadmin**

Exécute n'importe quelle opération sur le serveur, c'est l'administrateur du serveur.

**serveradmin**

Permet de configurer les paramètres au niveau du serveur.

**setupadmin**

Permet d'ajouter/supprimer des serveurs liés et d'exécuter certaines procédures stockées système telles que **sp_serveroptions**.

**securityadmin**

Permet de gérer les connexions d'accès au serveur.

**processadmin**

Permet de gérer les traitements s'exécutant sur SQL Server.

**dbcreator**

Permet de créer et modifier les bases de données.

**diskadmin**

Permet de gérer les fichiers sur le disque.

**bulkadmin**

Peut exécuter l'instruction BULK INSERT pour une insertion des données par blocs. L'intérêt de ce type d'insertion réside dans le fait que l'opération n'est pas journalisée et donc les temps d'insertion sont beaucoup plus rapides.

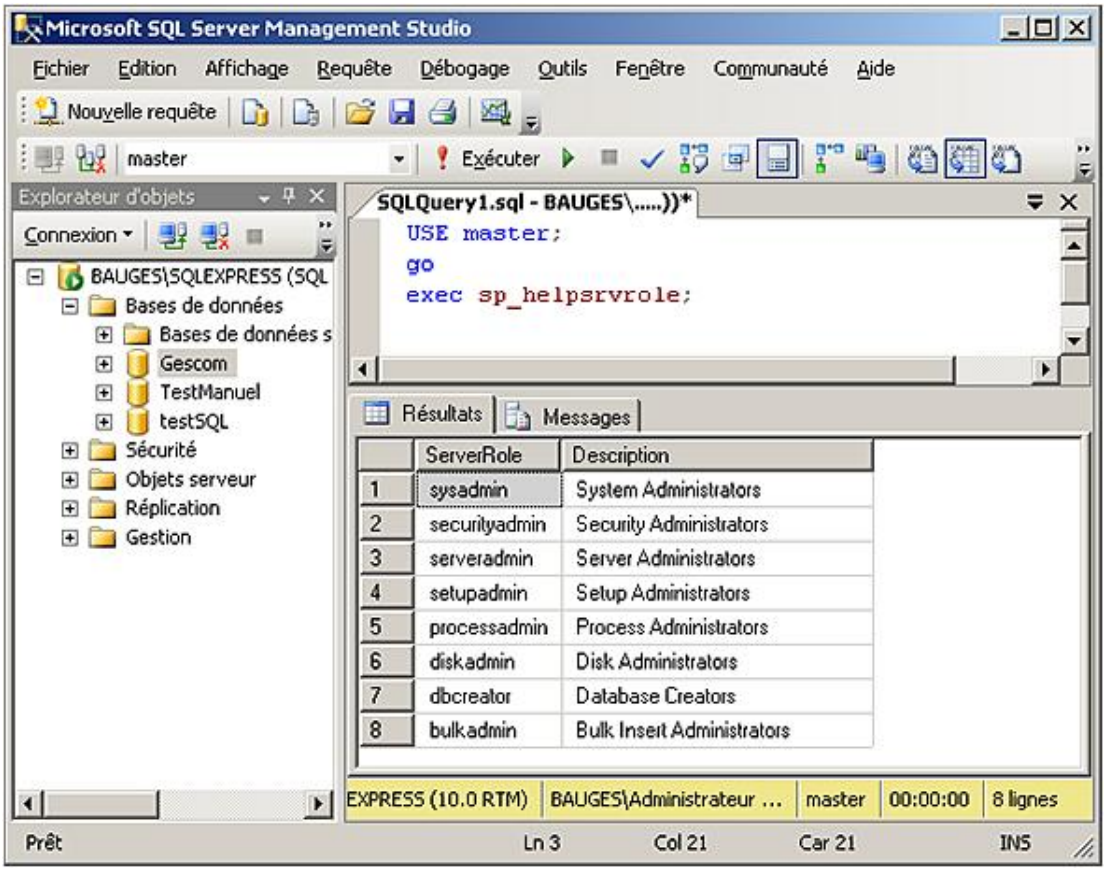
---

Seuls les membres du rôle **sysadmin** peuvent accorder un rôle de serveur à une connexion.

---

Le rôle **public** reçoit le privilège VIEW ANY DATABASE. Ainsi, les utilisateurs qui se connectent à SQL peuvent lister les bases définies sur l'instance. Ils ne pourront y accéder que si un compte d'utilisateur de base de données est mappé à leur connexion ou bien si l'utilisateur **guest** (invité) est défini au niveau de la base.

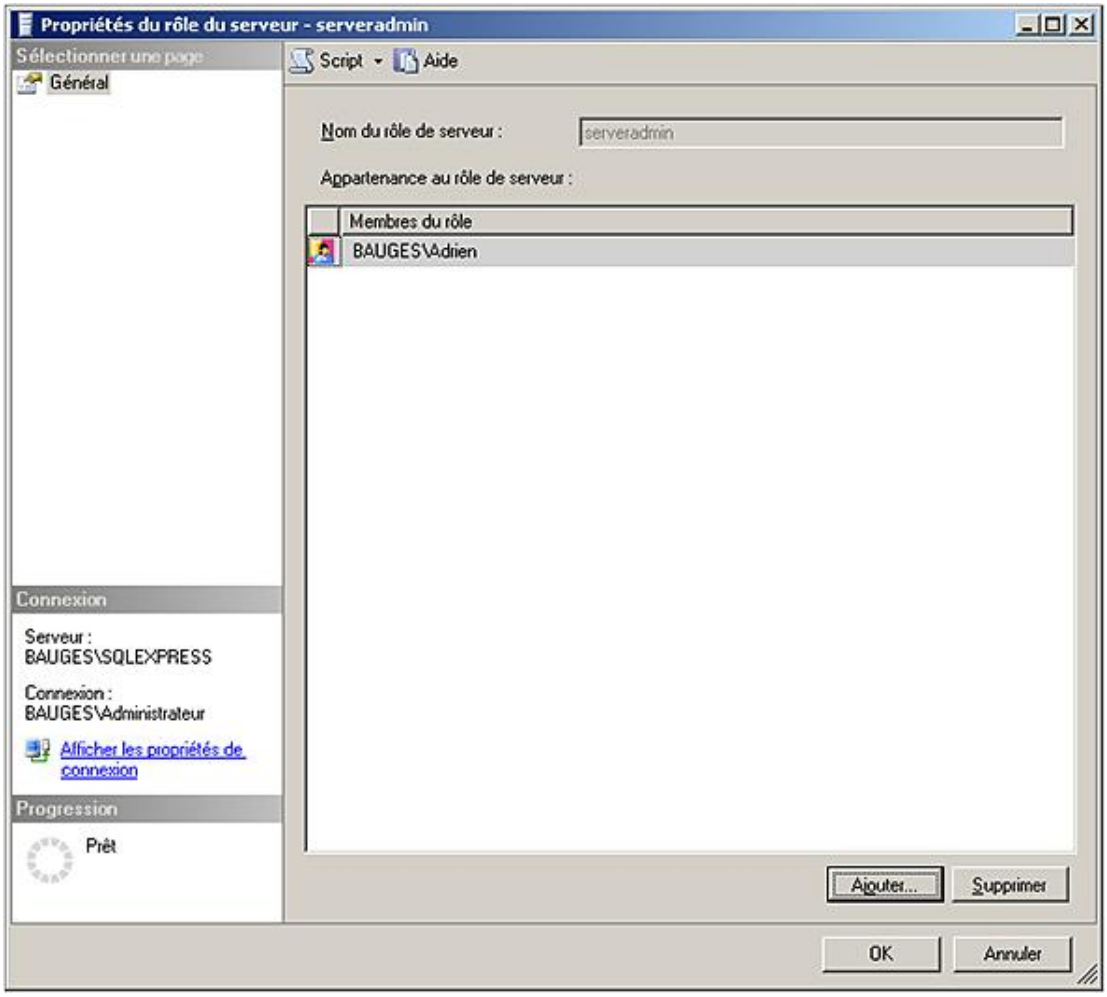
Les procédures **sp_helpsrvrole** et **sp_helpsrvrolemenber** permettent d'obtenir toutes les informations nécessaires sur les différents rôles fixes de serveur et leur attribution.




*Exécution de sp_helpsrvrole pour connaître les rôles de serveurs fixes*

## SQL Server Management Studio

La gestion des rôles fixes de serveur et plus exactement la gestion des connexions au sein du rôle est gérée depuis la fenêtre des propriétés du rôle.




## Transact SQL

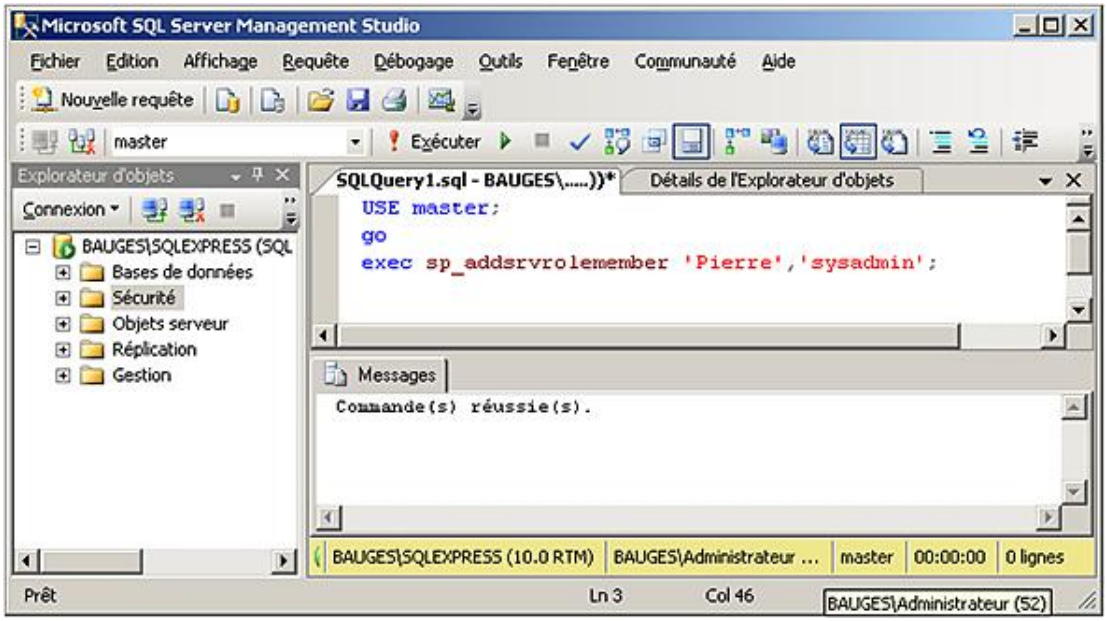
C'est par l'intermédiaire de deux procédures stockées que s'effectuent toutes les opérations.

### Ajouter un membre

*Syntaxe :*

```
sp_addsrvrolemember 'nom_connexion', 'nom_rôle'
```

*Exemple :*



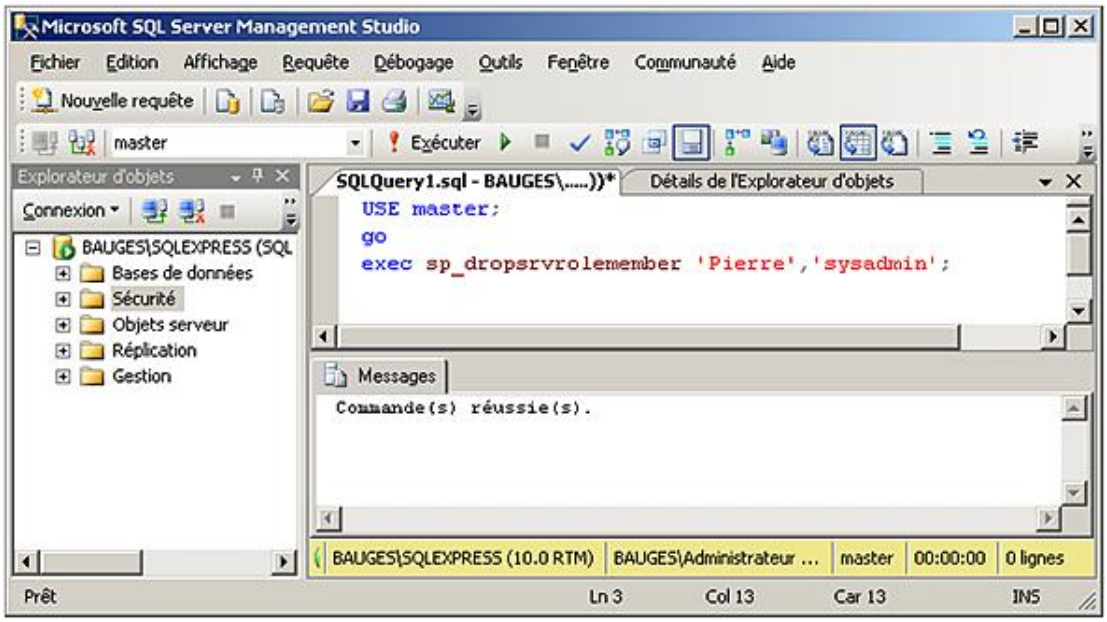

La connexion *Pierre* devient administrateur, c'est-à-dire membre du rôle de serveur **sysadmin**.

**Retirer un membre**

*Syntaxe :*

```
sp_dropsrvrolemember 'nom_connexion', 'nom_rôle'
```

*Exemple :*




## 2. Rôles de base de données

Les rôles de bases de données permettent toujours de regrouper les différentes autorisations ou refus d'instructions ou d'objets et ainsi de faciliter la gestion des droits. Il existe des rôles prédéfinis et il est possible de définir ses propres rôles.

Les modifications sur les rôles sont dynamiques et les utilisateurs n'ont pas besoin de se déconnecter/reconnecter de la base pour bénéficier des changements.

Les rôles prédéfinis ne peuvent pas être supprimés.

Seuls les membres des rôles fixes de base de données **db_owner** et **db_securityadmin** sont à même de gérer l'appartenance des utilisateurs à un rôle fixe ou non de la base de données.

### a. Le rôle public

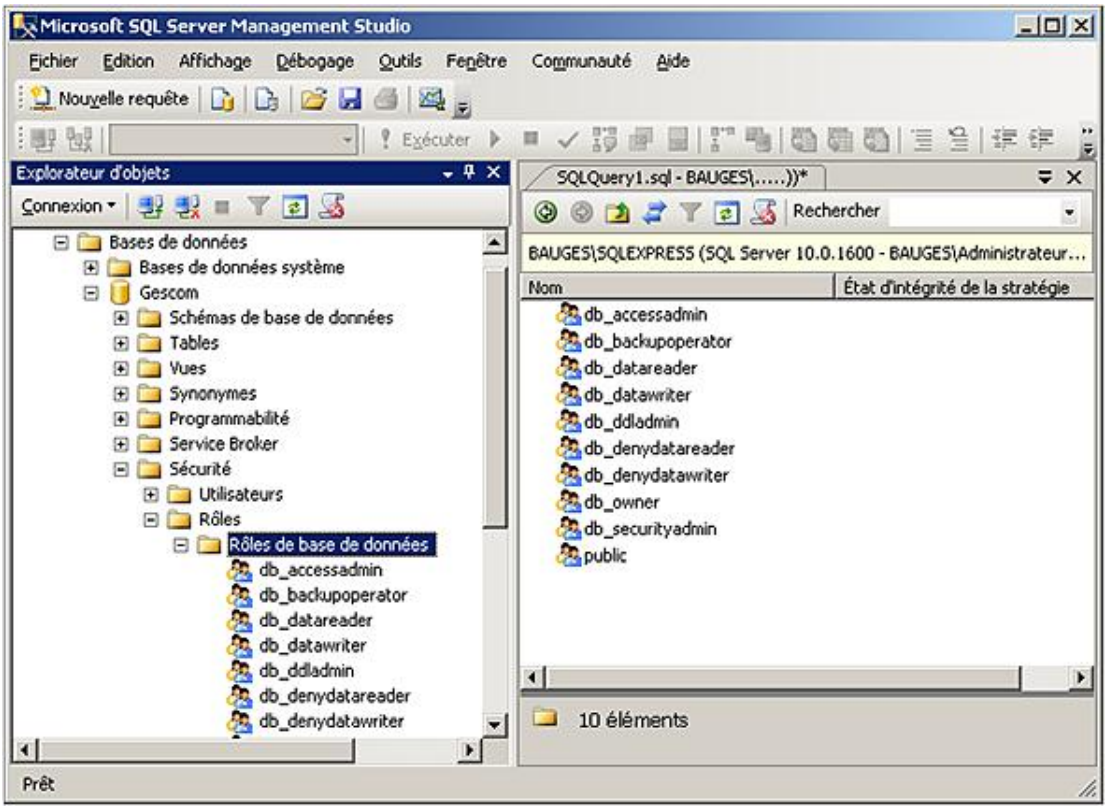
Il s'agit d'un rôle prédéfini, bien particulier, car tous les utilisateurs de la base possèdent ce rôle. Ainsi, lorsque qu'une autorisation d'instruction ou d'objet est accordée, supprimée ou interdite à **public**, instantanément tous les utilisateurs de la base bénéficient des modifications.

Le rôle **public** contient toutes les autorisations par défaut dont bénéficient les utilisateurs de la base.

Il n'est pas possible d'attribuer des membres à ce rôle puisque tout le monde le possède implicitement. Ce rôle est présent dans toutes les bases du serveur, aussi bien les bases système que les bases utilisateur.

À sa création, un utilisateur ne dispose d'aucune autorisation sauf celles accordées à public.

### b. Les rôles prédéfinis




Mis à part le rôle **public**, il existe des rôles prédéfinis dans la base de données.

**db_owner**

Ensemble de droits équivalents au propriétaire de la base. Ce rôle contient toutes les activités possibles dans les autres rôles de la base de données, ainsi que la possibilité d'exécuter certaines tâches de maintenance et de configuration de la base. Tous les membres de ce groupe vont créer des objets dont le propriétaire est **dbo.** Il s'agit du propriétaire par défaut de tous les objets et il n'est pas nécessaire de le préciser pour manipuler ses objets.

**db_accessadmin**

Permet d'ajouter ou de supprimer des utilisateurs dans la base de données. Ces utilisateurs correspondent à des connexions SQL Server ou bien à des groupes ou utilisateurs Windows.

**db_datareader**

Permet de consulter (SELECT) le contenu de toutes les tables de la base de données.

**db_datawriter**

Permet d'ajouter (INSERT), modifier (UPDATE) ou supprimer (DELETE) des données dans toutes les tables utilisateur de la base de données.

**db_ddladmin**

Permet d'ajouter (CREATE), modifier (ALTER) ou supprimer (DELETE) des objets de la base de données.

**db_securityadmin**

Permet de gérer les rôles, les membres des rôles, et les autorisations sur les instructions et les objets de la base de données.

**db_backupoperator**

Permet d'effectuer une sauvegarde de la base de données.

**db_denydatareader**

Interdit la visualisation des données de la base.

**db_denydatawriter**

Interdit la modification des données contenues dans la base.

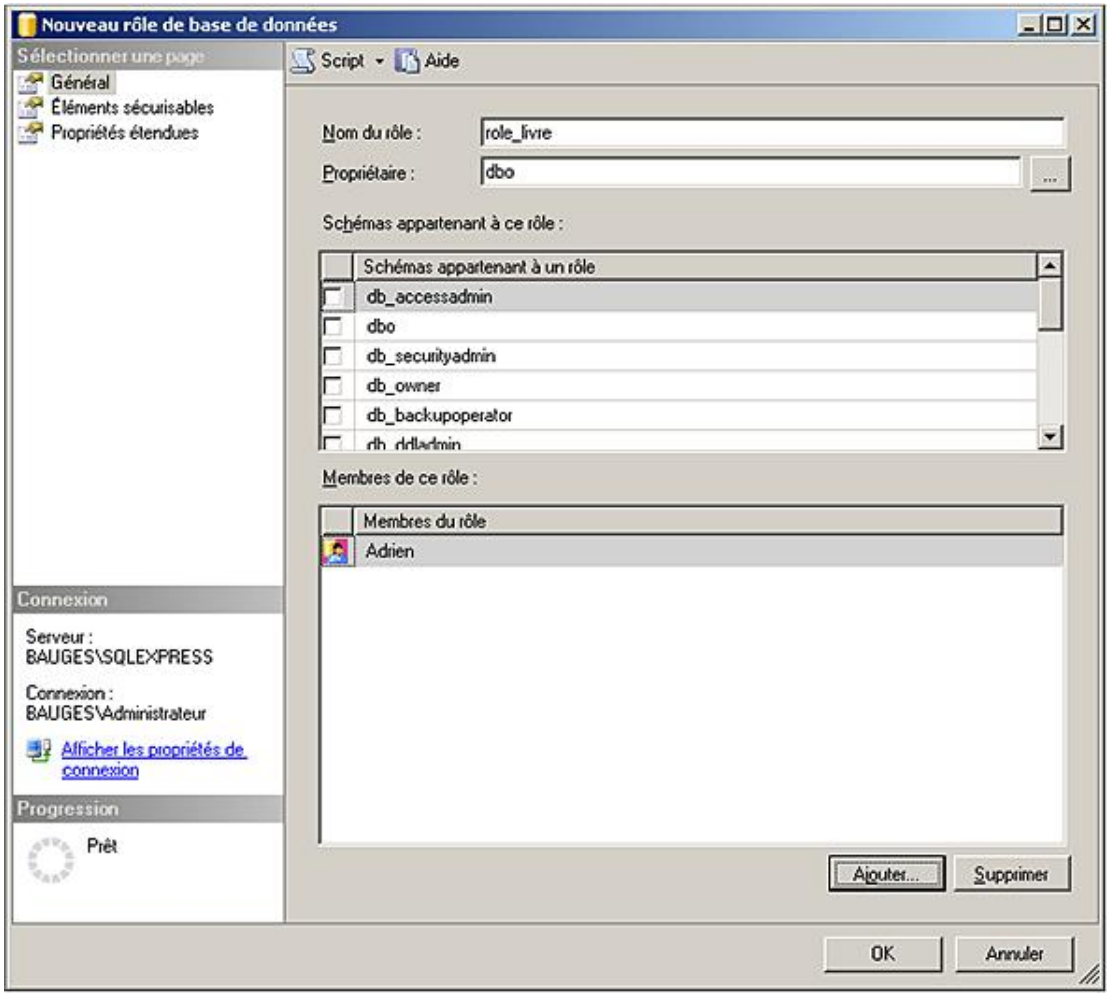
Tous les rôles prédéfinis sont présents dans toutes les bases système.

### c. Les rôles définis par les utilisateurs

Il est possible de définir ses propres rôles afin de faciliter l'administration des droits à l'intérieur de la base.

Le rôle SQL Server va être mis en place lorsque plusieurs utilisateurs Windows souhaitent effectuer les mêmes opérations dans la base et qu'il n'existe pas de groupe Windows correspondant ou bien lorsque des utilisateurs authentifiés par SQL Server et des utilisateurs authentifiés par Windows doivent partager les mêmes droits.

Pour les rôles définis au niveau base de données, il est possible de savoir quels utilisateurs en bénéficient, en demandant l'affichage de la fenêtre des propriétés du rôle depuis SQL Server Management Studio.

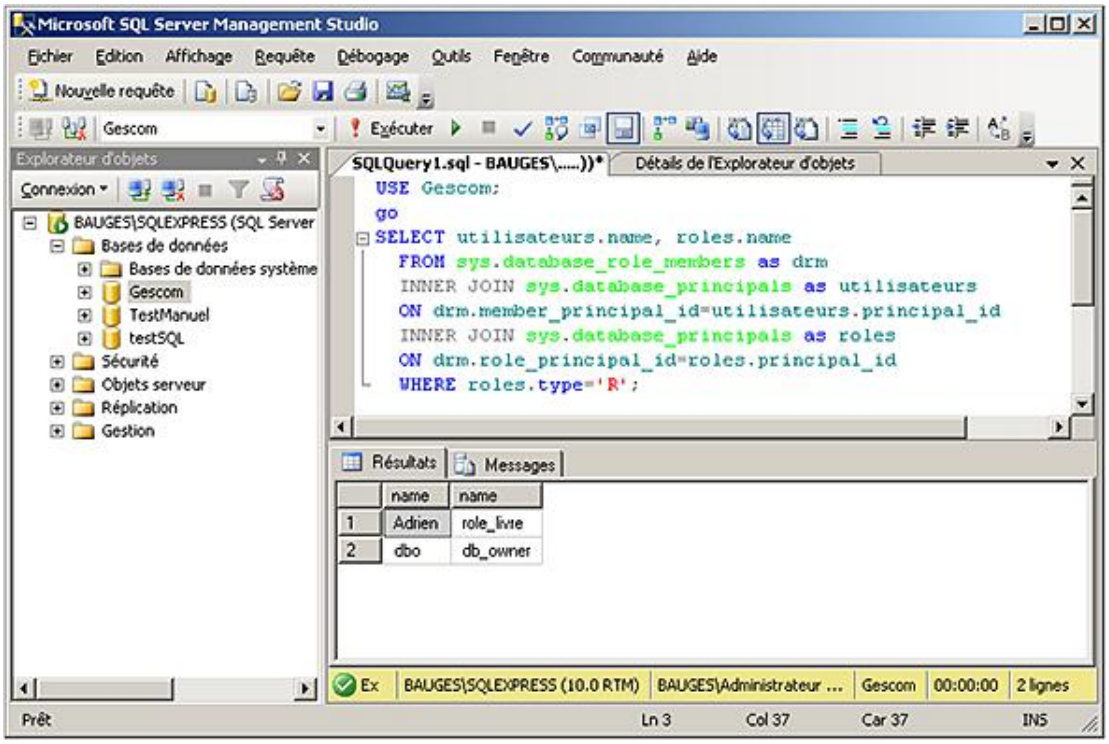



Les rôles peuvent être accordés soit directement à un utilisateur, soit à un autre rôle. Cependant, l'imbrication répétée des rôles peut nuire à la performance. De plus, il n'est pas possible de créer des rôles récursifs.

L'utilisation de rôles peut paraître parfois lourde lors de la création mais facilite grandement les évolutions qui peuvent intervenir (modification des autorisations, des utilisateurs).

Pour définir un rôle il faut être membre du rôle **sysadmin**, ou bien membre de **db_owner** ou **db_securityadmin**.

L'interrogation des tables système **sys.database_principals** et **sys.database_role_member** permet de connaître la liste des utilisateurs qui appartiennent à chaque rôle de base de données. L'exemple suivant illustre ce propos :




Comme l'ensemble des opérations d'administration, il est possible de gérer les rôles de deux façons différentes :

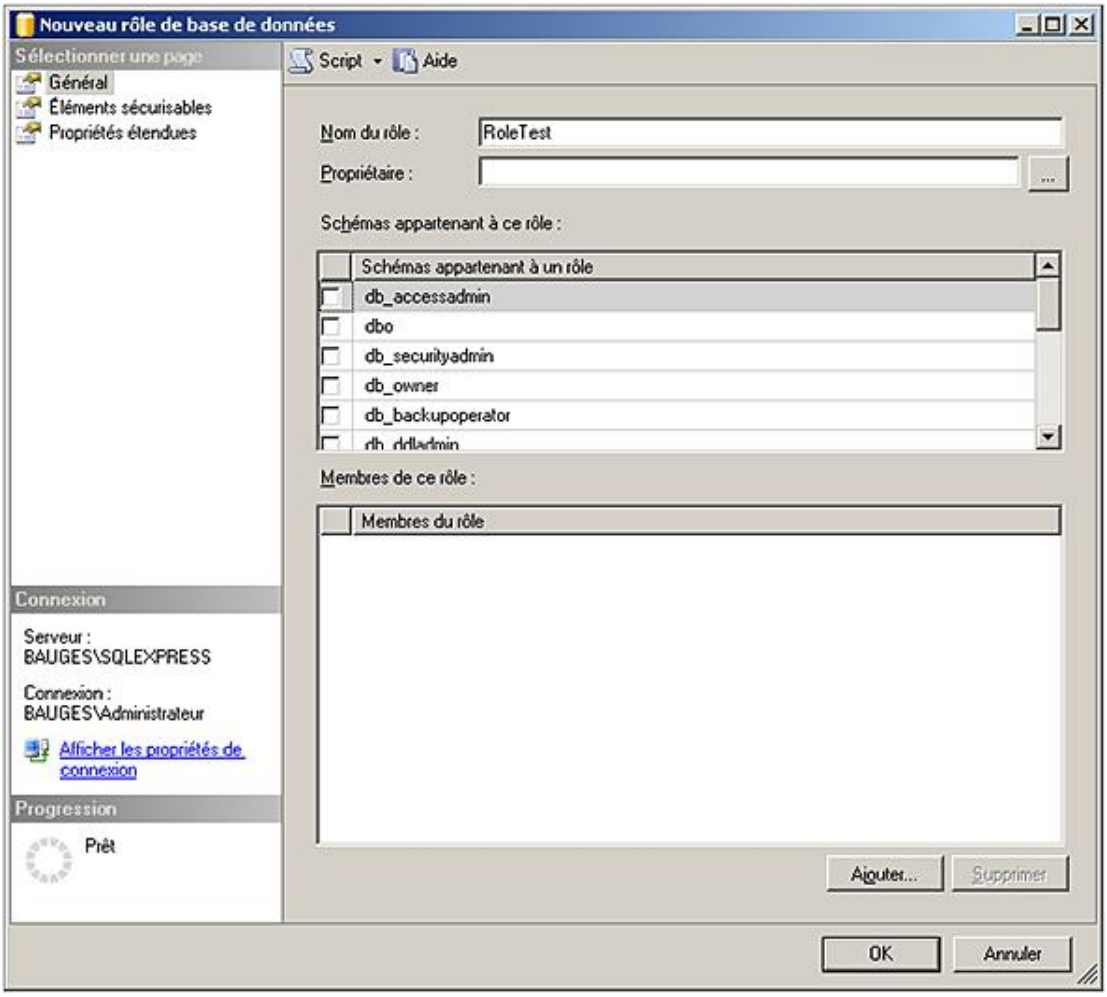
### SQL Server Management Studio

La création des rôles de base de données est possible en réalisant les opérations suivantes :

- Depuis l'explorateur d'objets, se positionner sur la base de données utilisateur concernée par cet ajout.
- Se positionner sur le nœud **Sécurité - Rôles - Rôles de base de données**.
- Depuis le menu contextuel associé au nœud **Rôles de base de données**, faire le choix **Nouveau rôle de base de données**.

*Exemple*

*Dans l'écran suivant, le rôle RoleTest est créé.*




C'est par l'intermédiaire de la fenêtre présentant les propriétés du rôle qu'il est possible de le modifier.

L'option de suppression est également disponible depuis le menu contextuel associé au rôle à supprimer.

**Transact SQL** **Création**

*Syntaxe :*

```
CREATE ROLE nomRole
[ AUTHORIZATION propriétaire ]
```

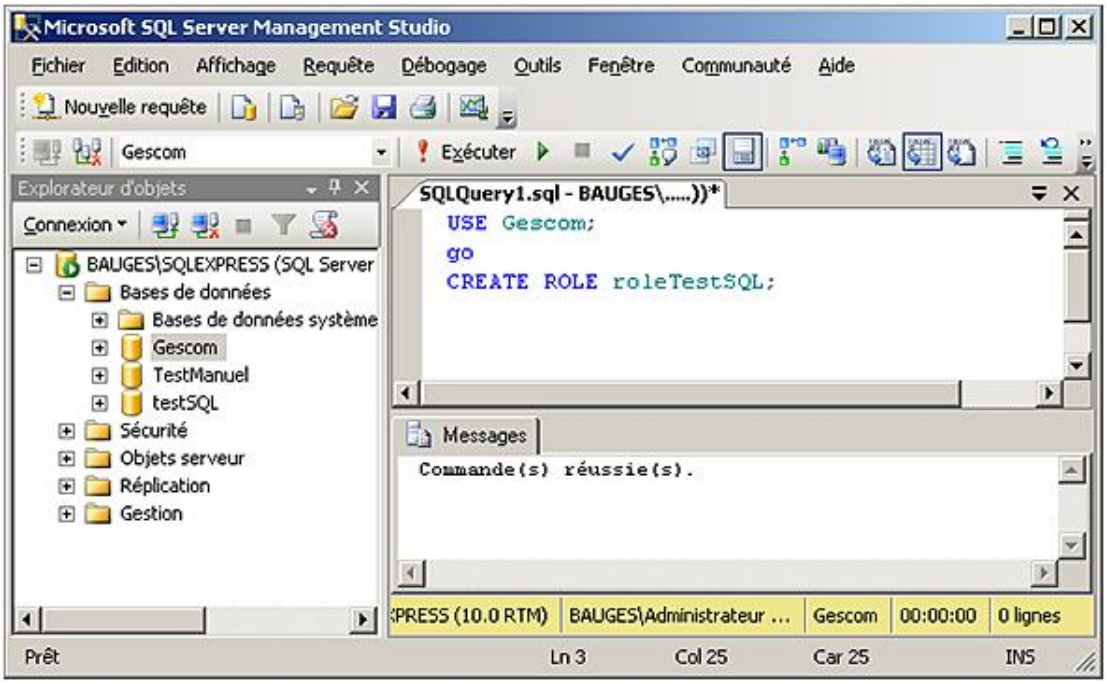
`nomRole`

Nom du rôle qui vient d'être créé.

`propriétaire`

Il est possible de préciser un propriétaire pour le rôle.

*Exemple :*




**Gestion des utilisateurs**

La gestion des membres du rôle se réalise en utilisant la procédure stockée **sp_addrolemember** pour ajouter un utilisateur.

*Syntaxe :*

```
sp_addrolemember 'nom_rôle',
'compte_sécurité'
```

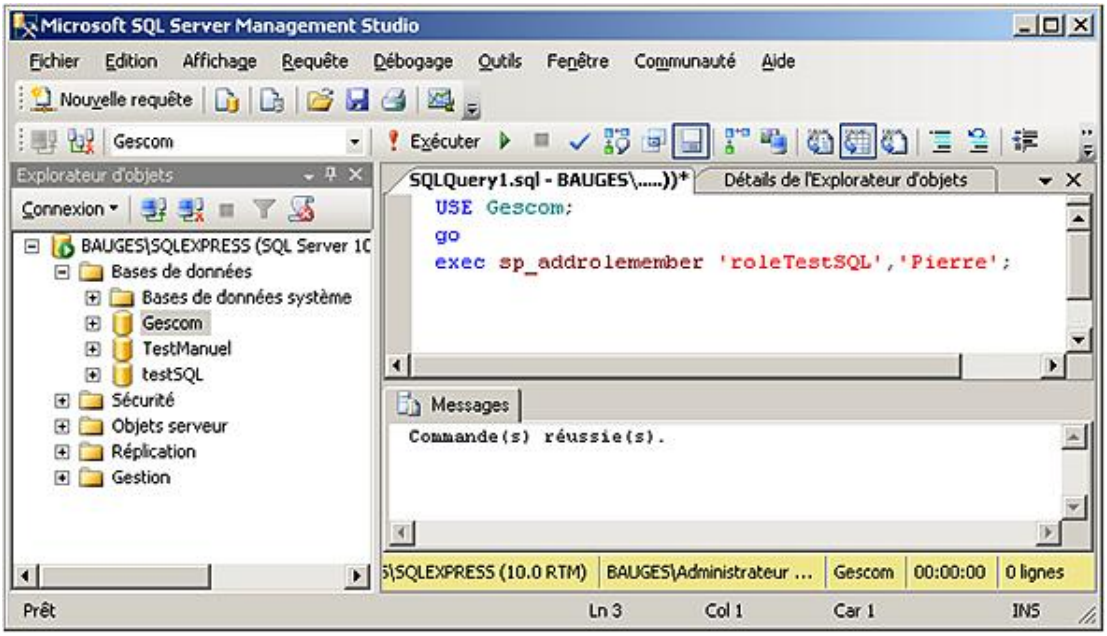
`Nom_rôle`

Nom du rôle auquel le compte d'utilisateur doit être ajouté.

`Compte_sécurité`

Nom d'un utilisateur de base de données ou bien d'un rôle SQL Server.

*Exemple :*




L'opération inverse s'effectue au moyen de la procédure **sp_droprolemember**.

*Syntaxe :*

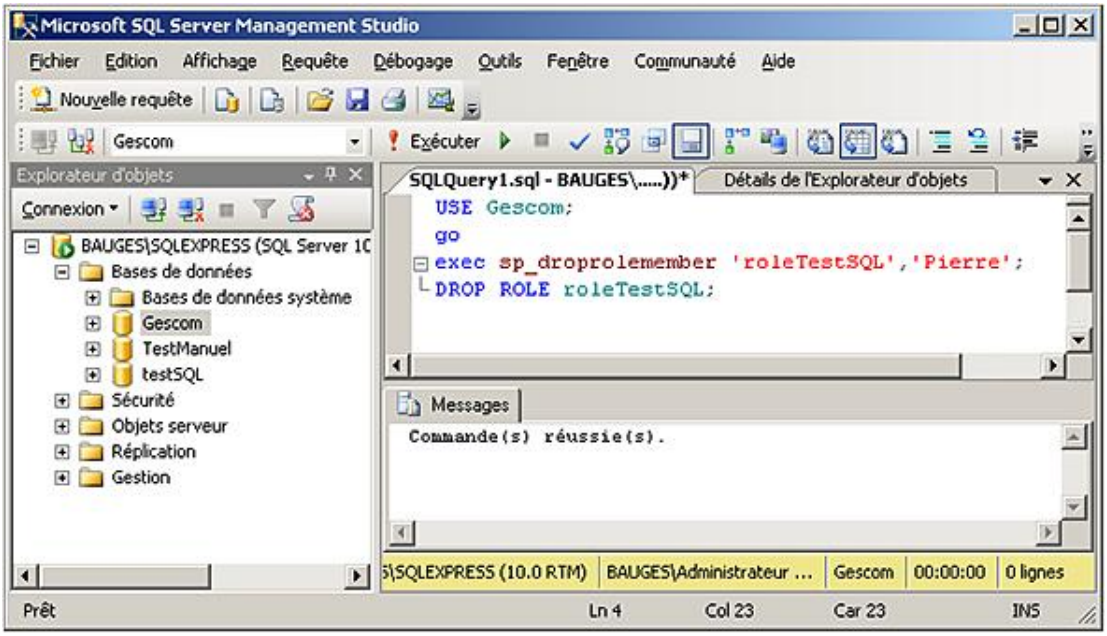
```
sp_droprolemember 'nom_role', 'compte_
sécurité'
```

**Suppression**

*Syntaxe :*

```
DROP ROLE nomRole
```

*Exemple :*




## 3. Rôles d'application

Les rôles d'application sont des rôles définis au niveau de la base de données sur laquelle ils portent. Comme tous les rôles, les rôles d'application permettent de regrouper des autorisations d'objets et d'instructions. Cependant les rôles d'application se distinguent car ils ne possèdent aucun utilisateur et ils sont protégés par un mot de passe.

La logique d'un rôle d'application est de permettre à tous les utilisateurs d'une application de posséder suffisamment de droits pour le bon déroulement de l'application où les opérations qui peuvent être réalisées sur la base de données sont contrôlées par le programme client. Cependant les utilisateurs eux-mêmes ne disposent pas de suffisamment de privilèges pour réaliser l'ensemble des opérations directement en SQL.

Lorsqu'un rôle d'application est activé depuis un applicatif client ou bien un script, les autorisations contenues dans ce rôle prennent le pas sur toutes les autorisations accordées directement ou par l'intermédiaire de rôles à l'utilisateur de la base de données.

Les rôles d'application permettent d'obtenir un comportement standard de l'application quel que soit l'utilisateur Windows qui lance l'application.

Comme les rôles d'application sont définis au niveau base de données, il n'est pas possible de leur accorder des privilèges sur d'autres bases. En effet, la connexion aux autres bases n'est possible que par l'intermédiaire du compte **guest**.

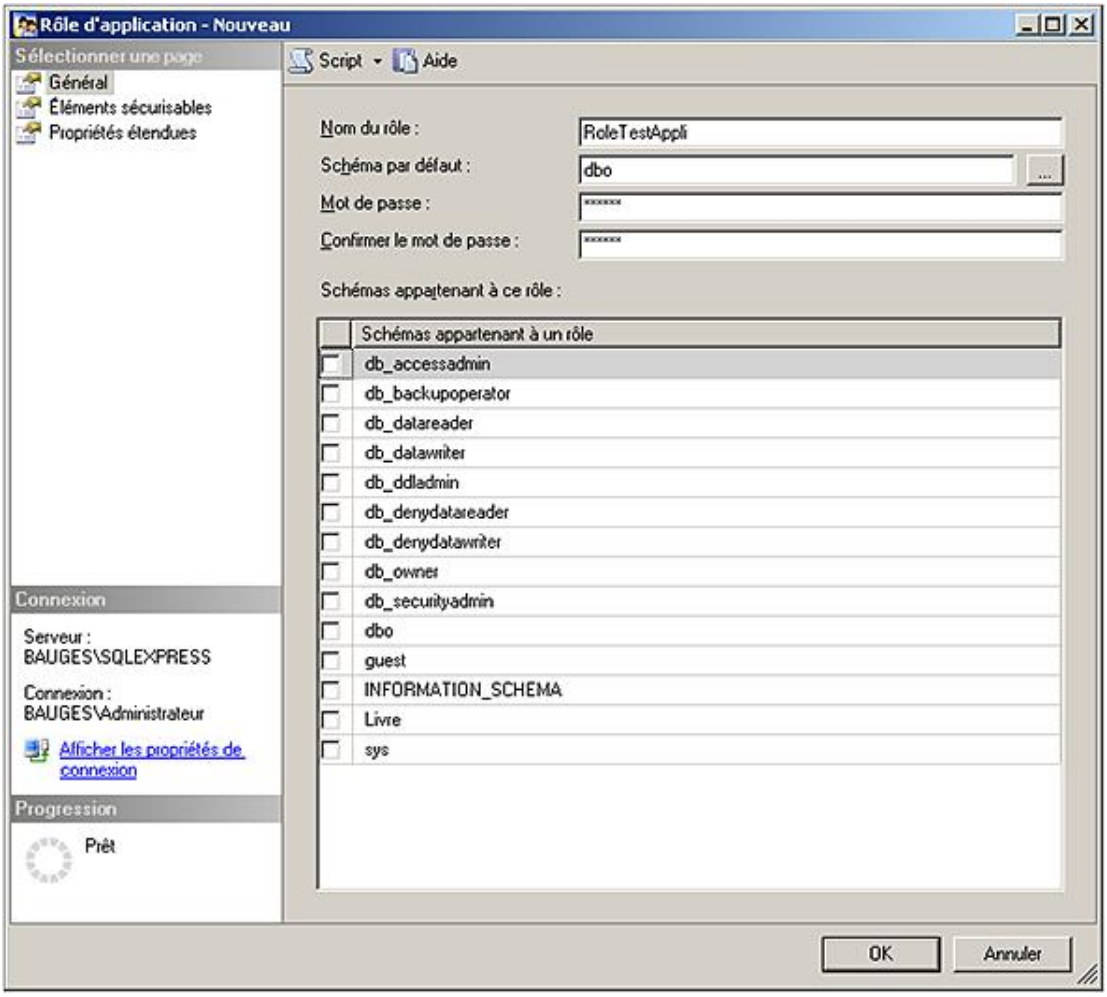
### a. SQL Server Management Studio

Les rôles d'application sont gérés de façon similaire à celle des rôles de base de données.

- Depuis l'explorateur d'objets, se positionner sur la base de données utilisateur concernée par cet ajout.
- Se positionner sur le nœud **Sécurité - Rôles - Rôles d'application**.
- Depuis le menu contextuel associé au nœud **Rôles d'application**, faire le choix **Nouveau rôle d'application**.

*Exemple*

*Le rôle d'application RoleTestAppli est créé.*




C'est par l'intermédiaire des **Propriétés du rôle** qu'il est possible de modifier le mot de passe du rôle.

L'ajout de permission sur des instructions ou des objets se réalise avec les mêmes étapes que la gestion des droits au niveau des utilisateurs.

La suppression d'un rôle de serveur se fait par l'intermédiaire du menu contextuel associé au rôle, en sélectionnant le choix **Supprimer**.

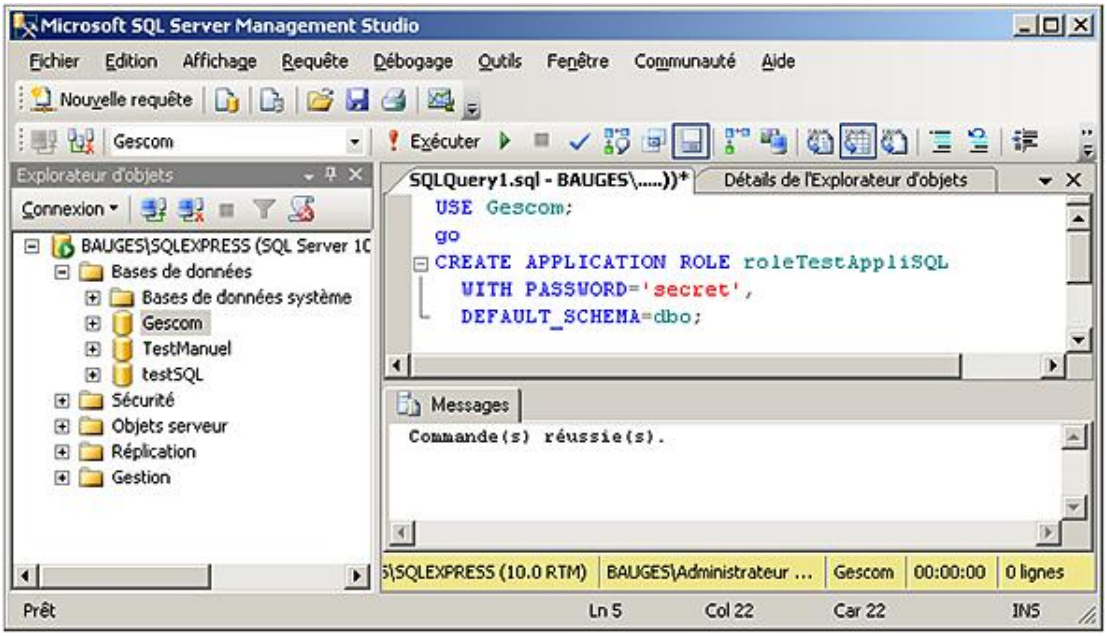
## b. Transact SQL

### Créer un rôle d'application

*Syntaxe*

```
CREATE APPLICATION ROLE nomRoleApplication
       WITH PASSWORD = 'motDePasse'
       [, DEFAULT_SCHEMA = schéma ]
```

*Exemple :*

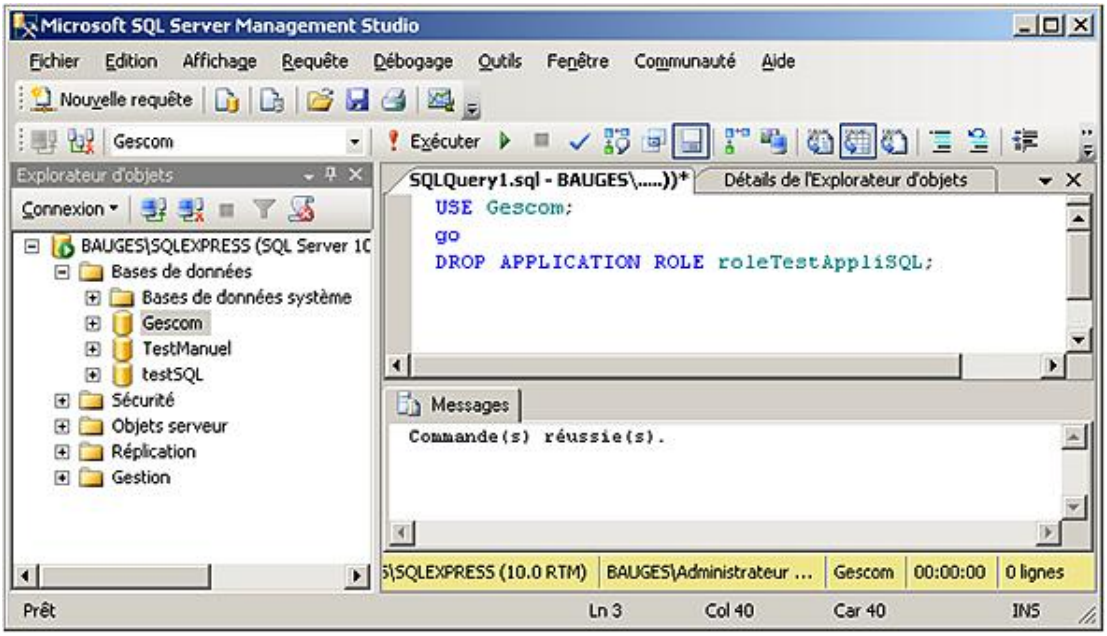



## Supprimer un rôle d'application

*Syntaxe*

```
DROP APPLICATION ROLE nomRoleApplication
```

*Exemple :*

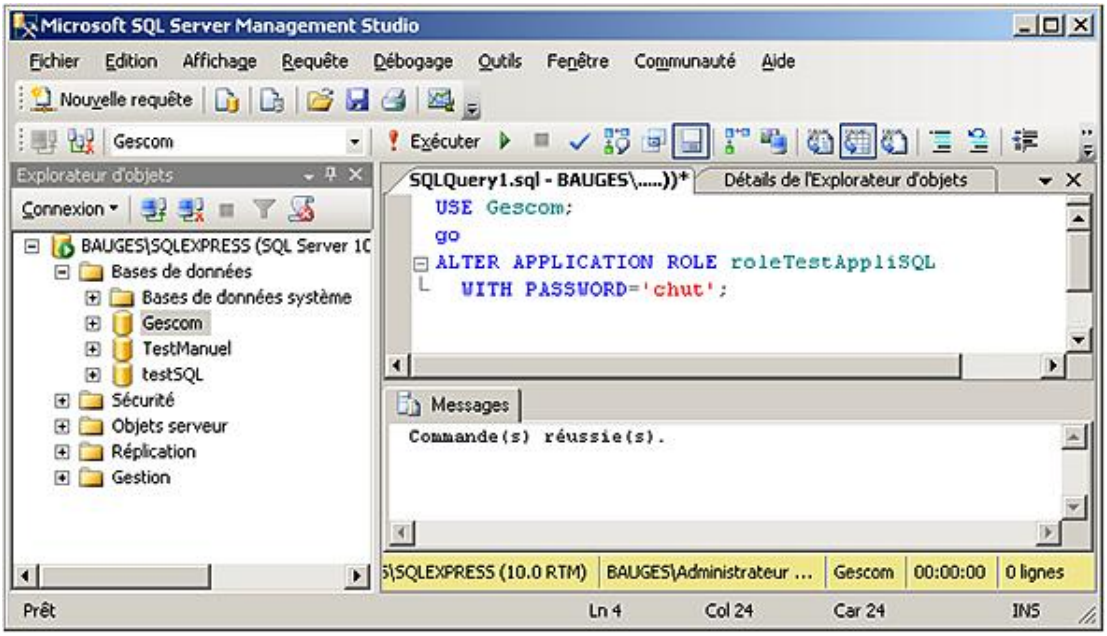



## Modifier le rôle

L'instruction ALTER permet de modifier le mot de passe mais également le schéma par défaut et le nom du rôle.

```
ALTER APPLICATION ROLE nomRoleApplication
        WITH {NAME = nouveauNom|
              PASSWORD = 'nouveauMotDePasse'|
              DEFAULT_SCHEMA = nouveauNomSchéma}
```

*Exemple :*




### c. L'utilisation

L'activation d'un rôle d'application se fait au moyen de la procédure stockée **sp_setapprole**. Dès qu'un rôle de serveur est activé, les droits de l'utilisateur sont ignorés et seules comptent les autorisations attribuées au rôle d'application.

*Syntaxe :*

```
sp_setapprole 'rôle',
[Encrypt N] 'mot-passe'
[, 'style_cryptage']
```

`rôle`

Nom du rôle d'application qui va être activé.

`mot_passe`

L'activation d'un rôle d'application est conditionnée par le mot de passe. Le mot de passe peut être crypté par l'intermédiaire de la fonction canonique ODBC Encrypt. L'option `N` permet de convertir le mot de passe au format unicode.

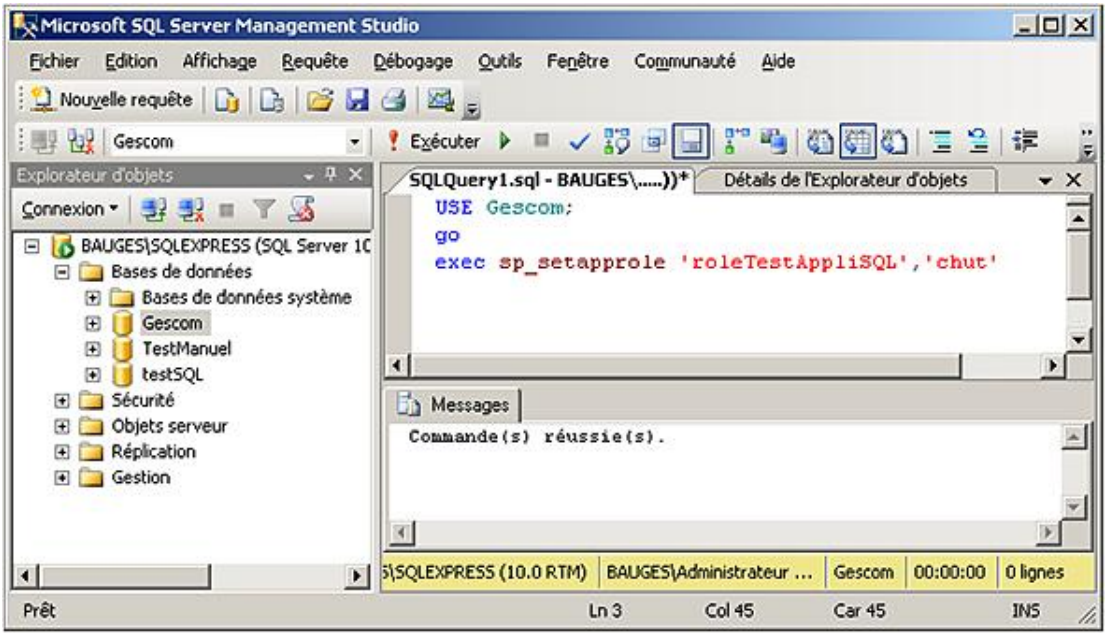
`style_cryptage`

Permet de spécifier un style de cryptage pour le mot de passe. Deux styles sont disponibles :

- none : le mot de passe est transmis à SQL Server sans codage préalable. C'est l'option par défaut.

- odbc : le mot de passe est codé à l'aide de la fonction Encrypt d'ODBC. Cette fonctionnalité n'est possible que sur un client ODBC communiquant avec le serveur OLE DB de SQL Server.

*Exemple :*



# 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 :*

```
ENIEditions, 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 :*

```
[ENI éditions], "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.

---

Le type sysname, qui est rencontré lorsque l'on travaille avec les tables système, est utilisé pour référencer les noms d'objets. Ce type est identique à un nvarchar(128) avec pour particularité que les valeurs null sont interdites.

---

### 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 $-10^{38}$ à $10^{38}$ -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 $-2^{63}$ (-9 223 372 036 854 775 808) et $2^{63}$-1(9 223 372 036 854 775 807).

`int`

Nombre entier entre $-2^{31}$ (-2147783648) et $+2^{31}$ -1 (+2147483647). Le type de données **int** est spécifique SQL Server et son synonyme **integer** est quant à lui compatible ISO.

`smallint`

Nombre entier entre $-2^{15}$ (-32768) et $2^{15}$ -1 (+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 -922 337 203 685 477, 5808 et +922 337 203 685 477, 5807 (8 octets).

`smallmoney`

Numérique au format monétaire compris entre -214 748,3648

et +214 748,3647 (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.

Le type `datetimeoffset` 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 d'un 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;
```

Résultats | Messages

| | 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)`.

SQL Server propose également un certain nombre de synonymes par rapport à ses propres types de base. Les

synonymes sont souvent présents pour assurer la compatibilité avec le standard ISO.

| Synonyme | Type SQL Server |
|---|---|
| Caractères<br>char varying<br>character(n)<br>character varying(n)<br>national character(n)<br>national char(n)<br>national character varying(n)<br>national char varying(n)<br>national text | varchar<br>char(n)<br>varchar(n)<br>nchar(n)<br>nchar(n)<br>nvarchar(n)<br>nvarchar(n)<br>ntext |
| Numériques<br>dec<br>double precision<br>integer | decimal<br>float<br>int |
| Binaires<br>binary varying | varbinary |
| Autres<br>rowversion | timestamp |

### 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 cet ouvrage.

**Syntaxe (suppression)**

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

Pour définir un type de données qui s'appuie sur un type du CLR, il faut activer la prise en charge du CLR par l'intermédiaire de **sp_dboption**.

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. :*

```
CREATE TYPE typenom
  FROM VARCHAR(30) NULL;
```

Messages

```
Commande(s) réussie(s).
```

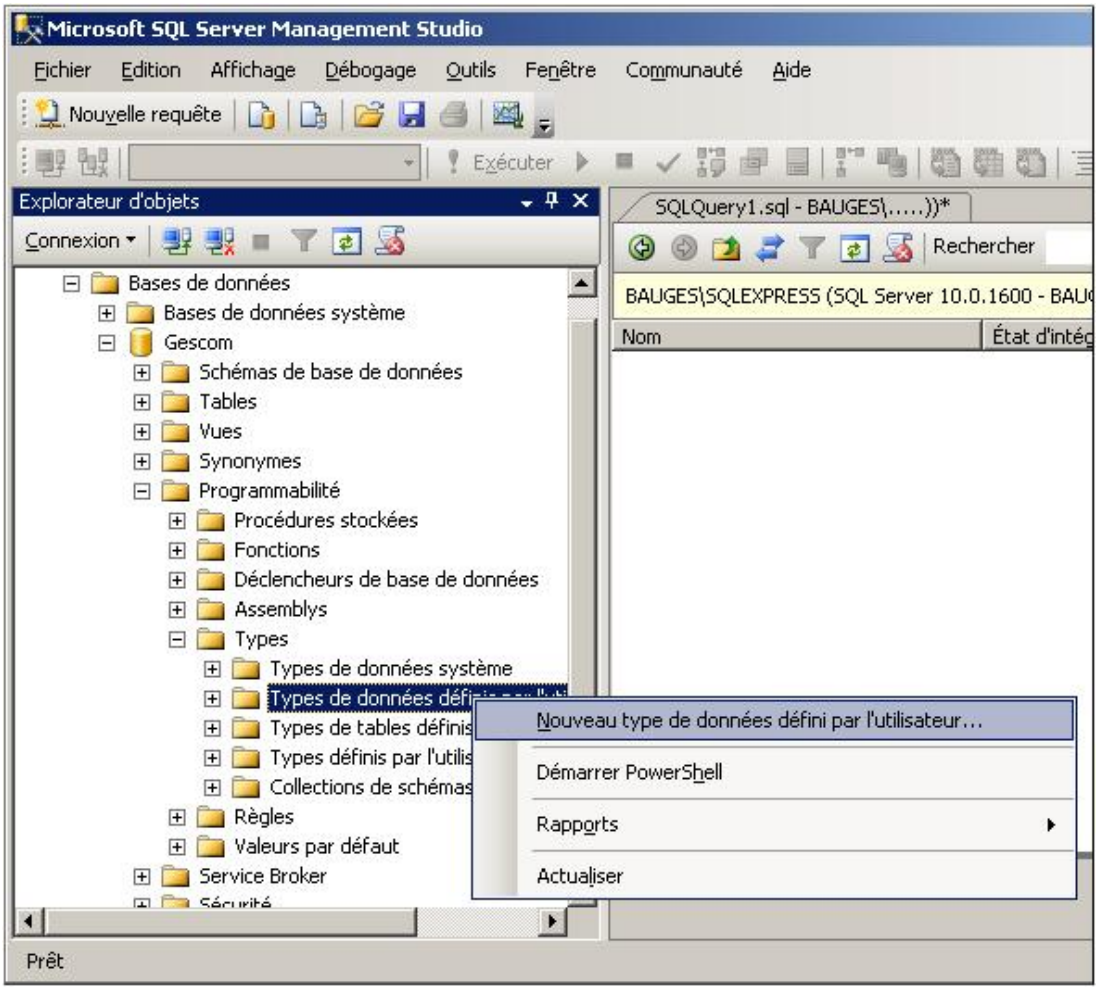
*Création d'un type pour des valeurs entre -999 et +999 :*

```
CREATE TYPE entier_3
  FROM NUMERIC(3) NOT NULL;
```

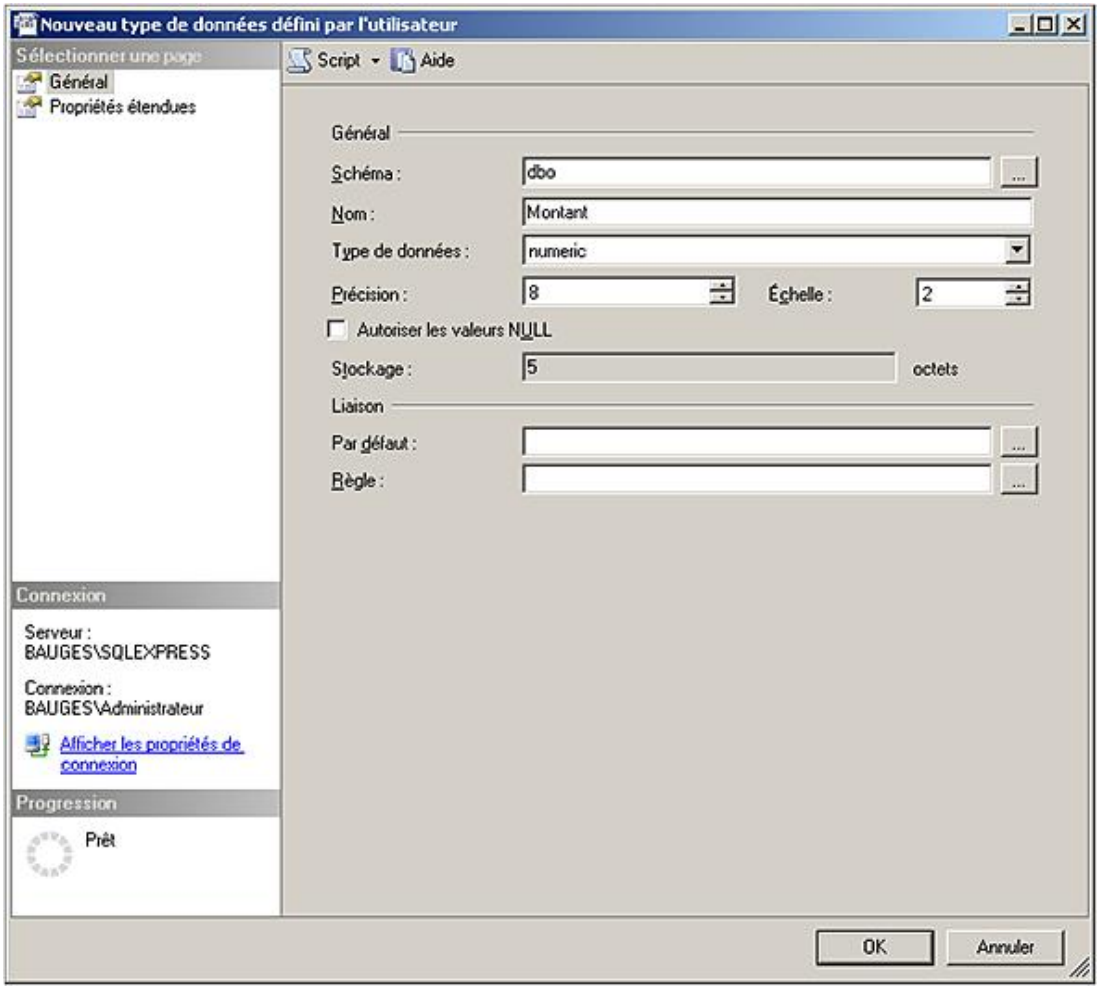
Messages

```
Commande(s) réussie(s).
```

*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 ci-dessus), 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 nomType AS TABLE (
colonne typeColonne[contrainteColonne], ...)
```

`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`

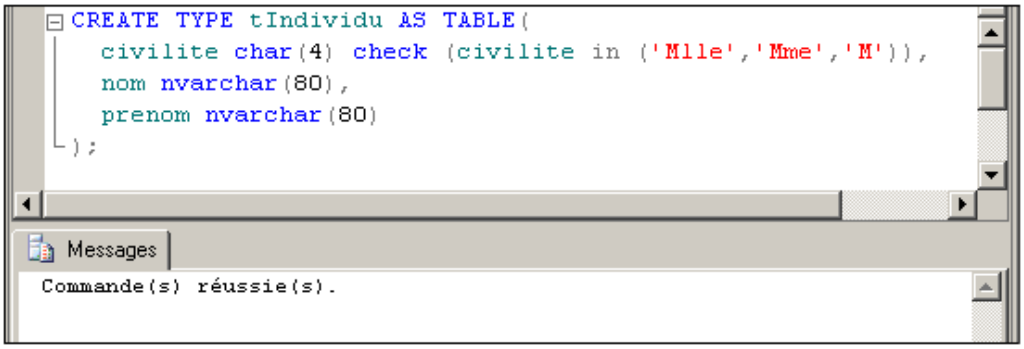
Type de données Transact SQL sur lequel est définie la colonne. Toutes les colonnes d'un même type de table ne sont pas forcément définies sur le même type ni avec la même précision.

`contrainteColonne`

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.*




## 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 [nomSchema.] nom_table
( nom_colonne {typecolonne|AS expression_calculé}
[,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 cet ouvrage).

`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.

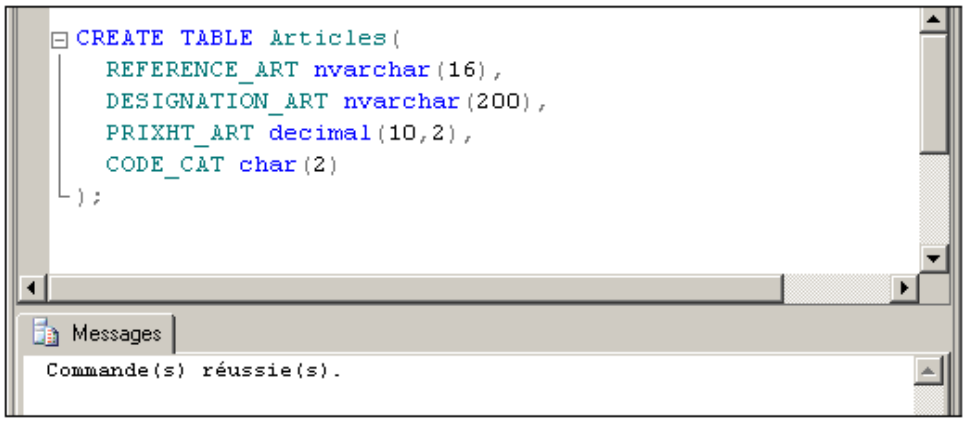
---

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).

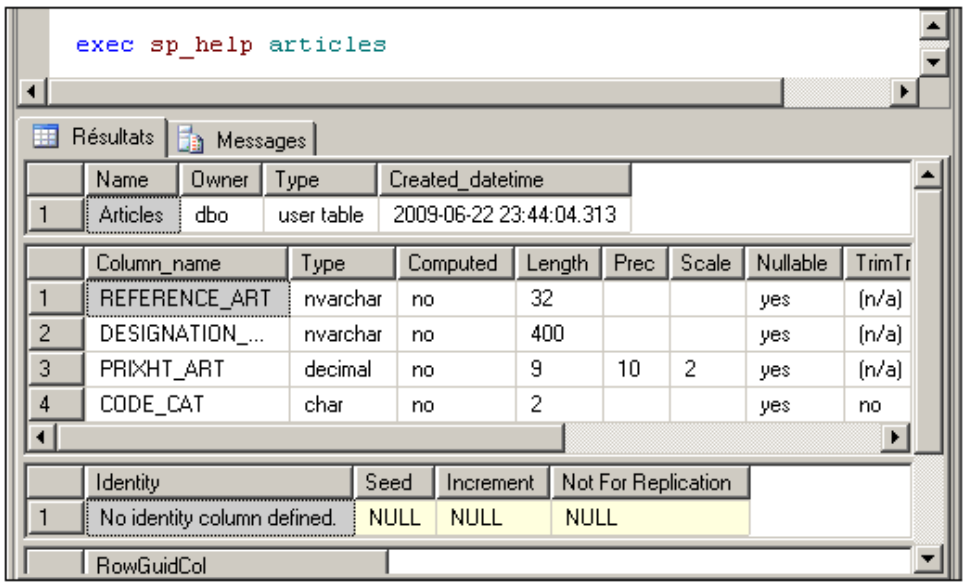
---

*Exemples*

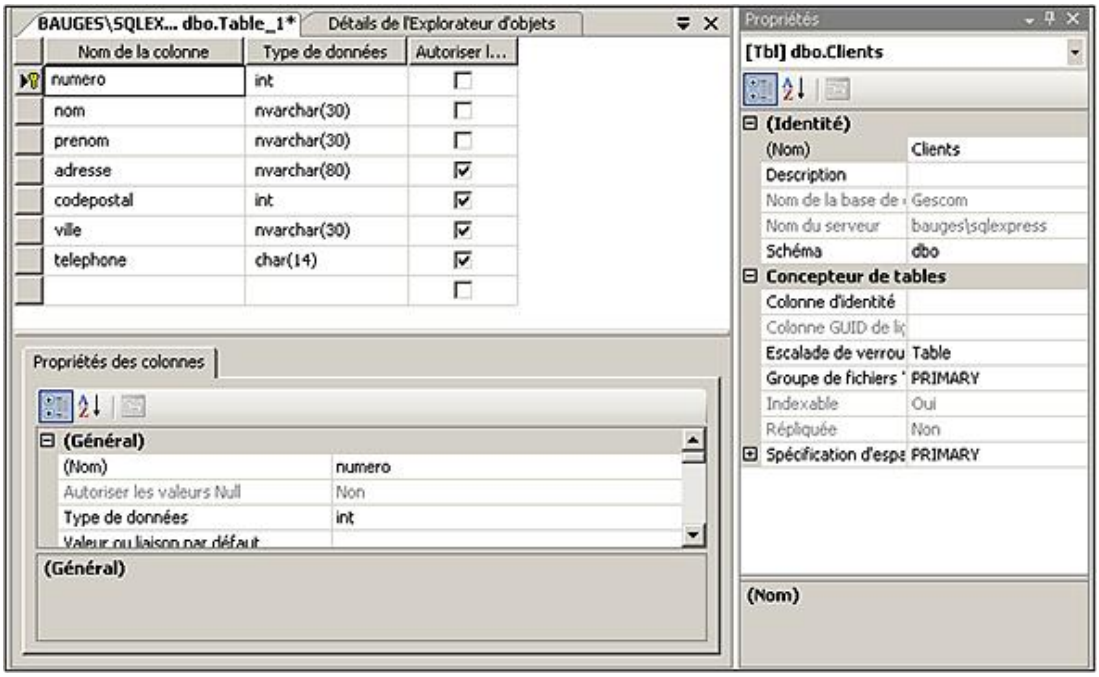
*Création de la table ARTICLES :*




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




*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`

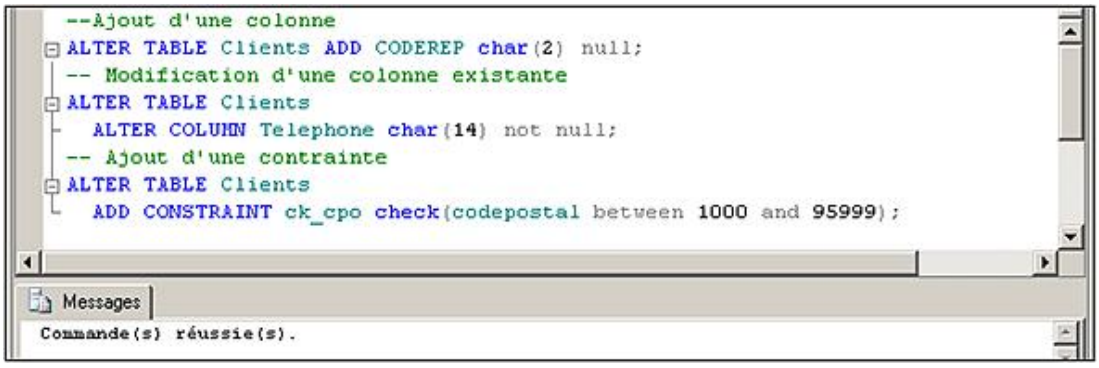
Permettent d'activer et de désactiver des contraintes d'intégrité.

`ENABLE, DISABLE`

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

*Exemple*

*Ajout d'une colonne :*




## c. Supprimer une table

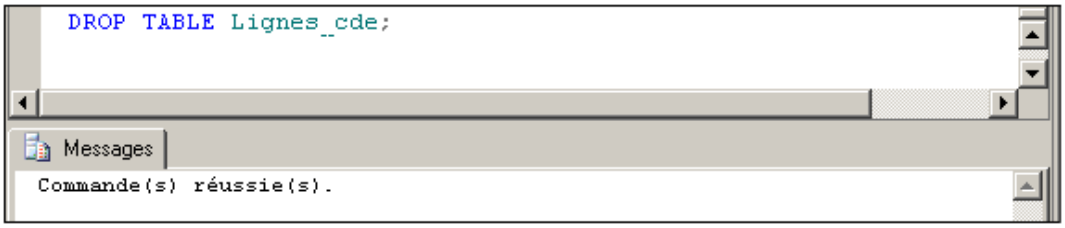
La suppression d'une table entraîne la suppression de toutes les données présentes dans la table. Les déclencheurs et les index associés à la table sont également supprimés. Il en est de même pour les permissions d'utilisation de la table. Par contre, les vues, procédures et fonctions qui référencent 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 œuvre 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 œuvre 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)],
 ...)
```

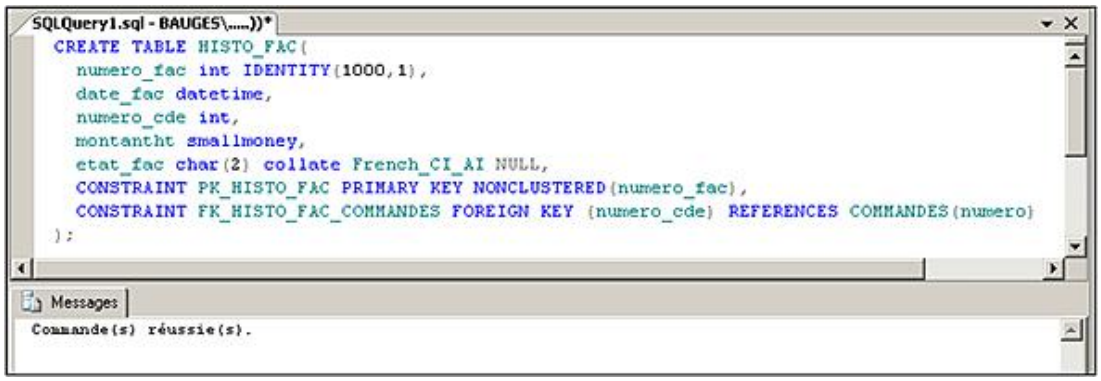
Il ne peut y avoir qu'une colonne IDENTITY par table !

---

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.

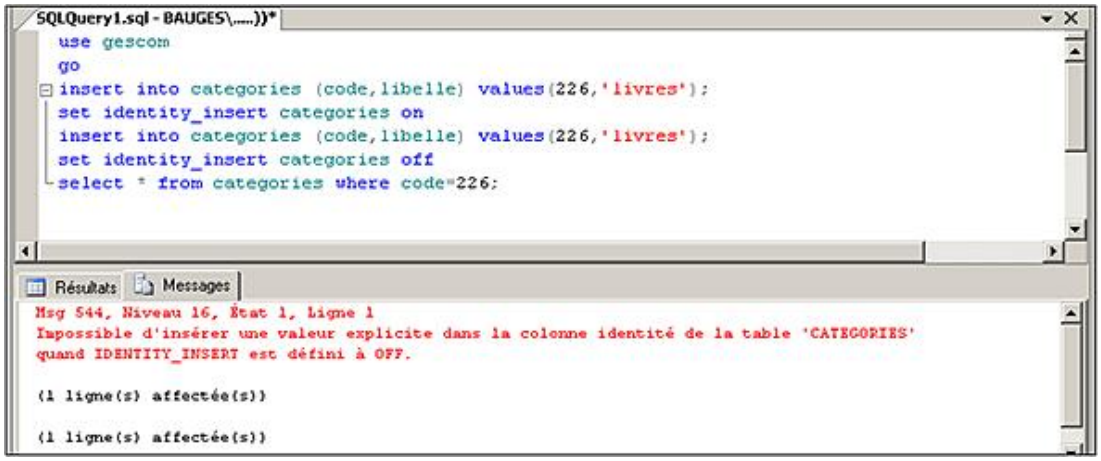
Pour pouvoir insérer des données sans utiliser la propriété IDENTITY et donc la numération automatique, il faut faire appel à l'instruction IDENTITY_ INSERT de la façon suivante :

```
SET IDENTITY_INSERT nom_de_table ON
```

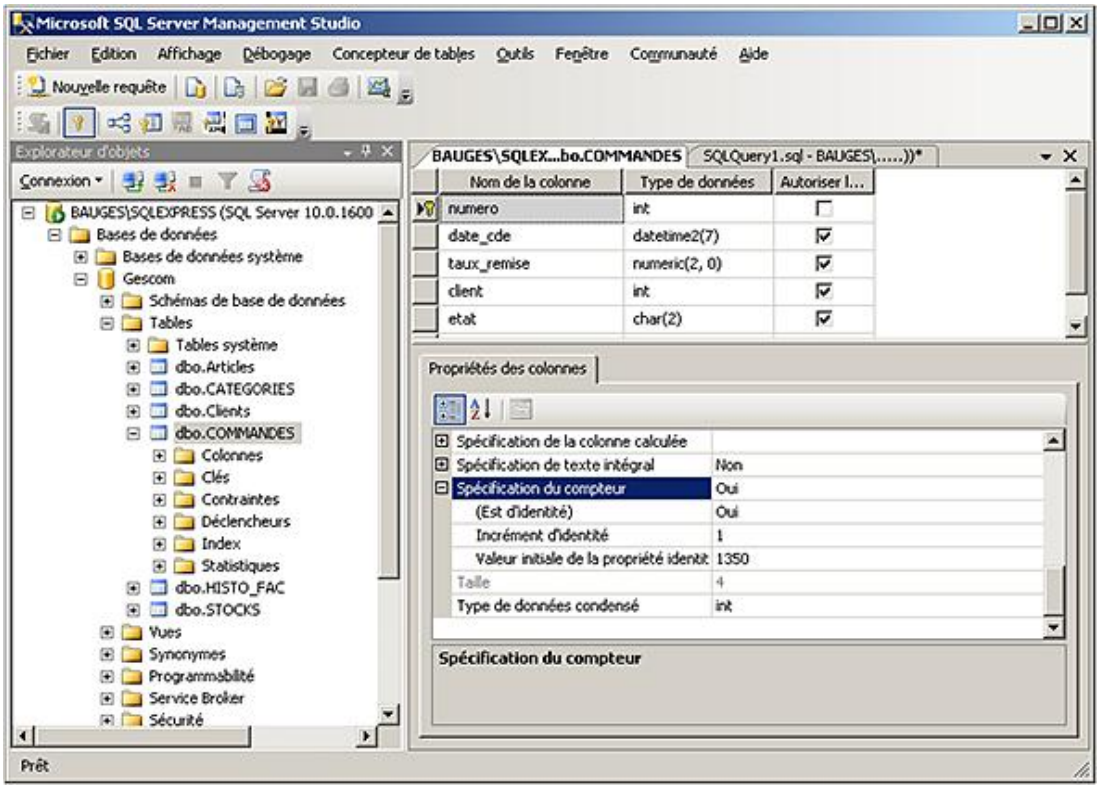
Le paramètre ON permet de désactiver l'usage de la propriété IDENTITY tandis que la même instruction avec le paramètre OFF réactive la propriété.

*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.*




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 (**Création** 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 œuvre 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 nomTable
{ADD|DROP} CONSTRAINT nomContrainte ...[;]
```

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 nomtable (nomcolonne 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 nomcontrainte]PRIMARY KEY[CLUSTERED|NONCLUSTERED]
(nomcolonne[,...])[WITH 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.

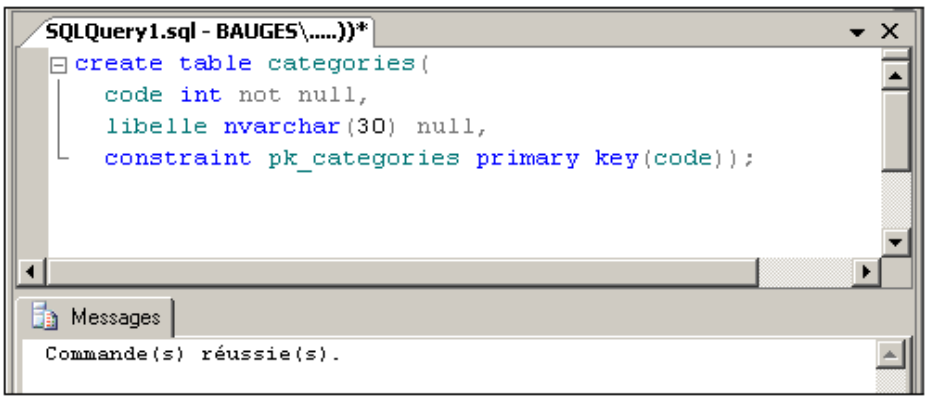
---

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.

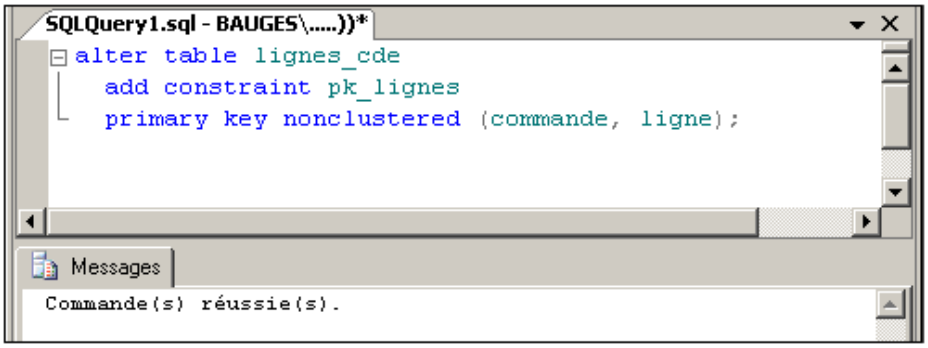
---

*Exemples*

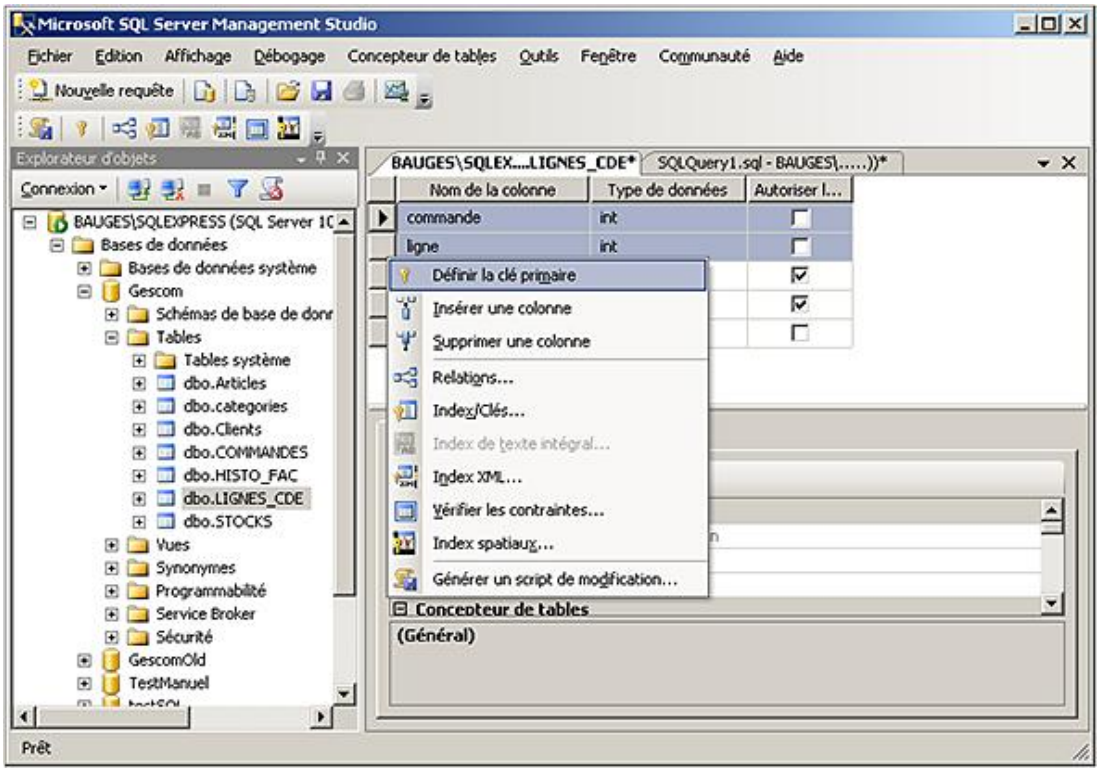
*Table catégorie, identifiant CODE_CAT :*



*Ajout de la clé primaire à la table Lignes_cdes (un index ordonné existe déjà sur Numero_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.

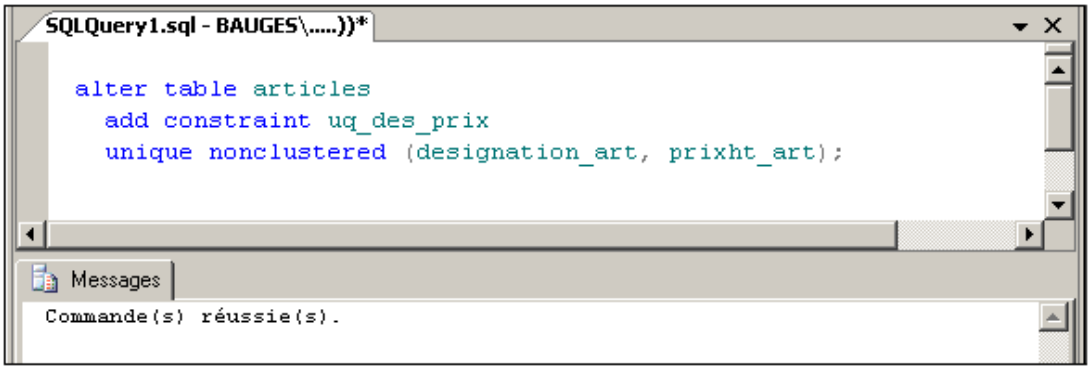
La gestion de cette contrainte est assurée par un index de type UNIQUE. Il n'est pas possible de supprimer cet index par l'intermédiaire de la commande DROP INDEX. Il faut supprimer la contrainte par l'intermédiaire de ALTER TABLE.

**Syntaxe**

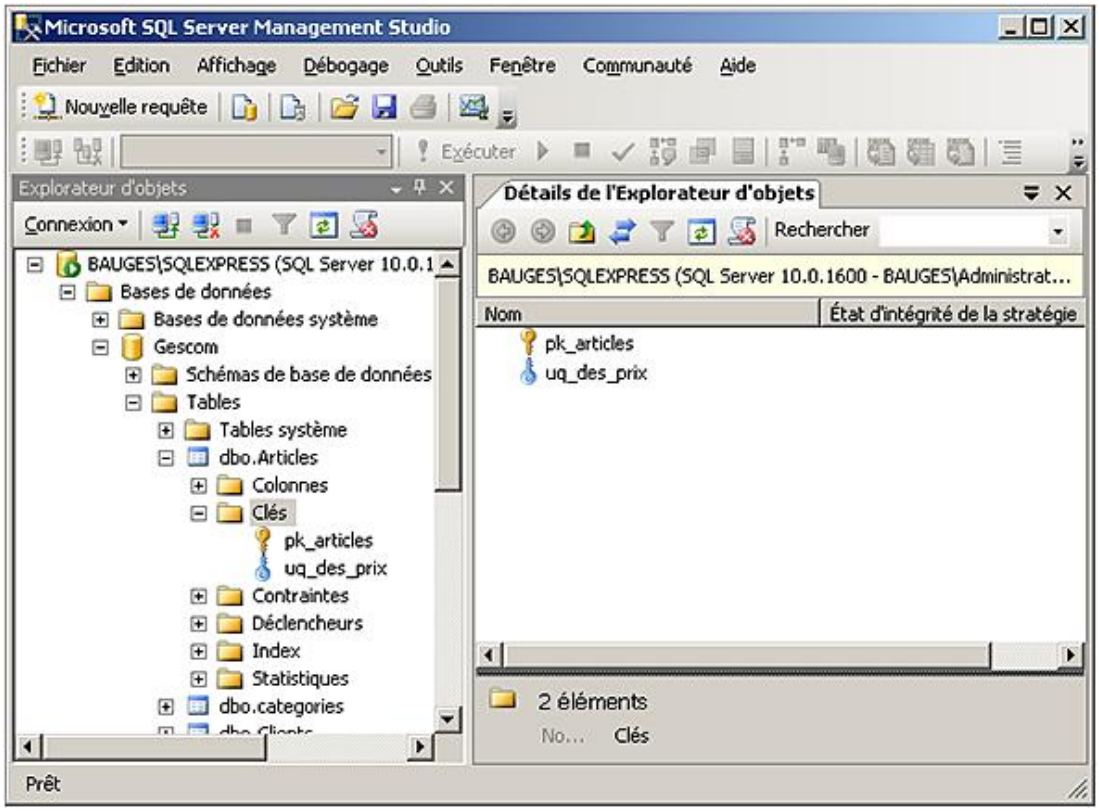
```
[CONSTRAINT nomcontrainte] UNIQUE [CLUSTERED | NONCLUSTERED ]
(nomcolonne [,...]) [WITH FILLFACTOR=x] [ON groupe_de_fichiers]
```

*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.

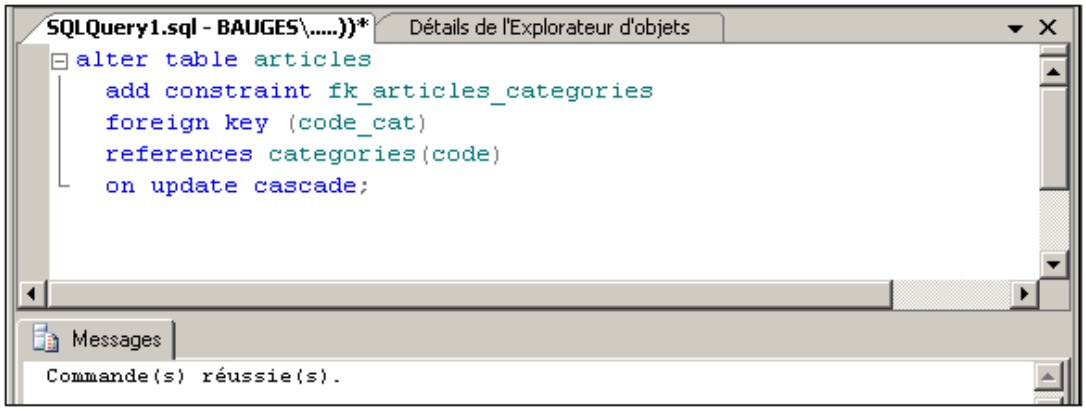
Toute référence circulaire est interdite dans les déclenchements des cascades.

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

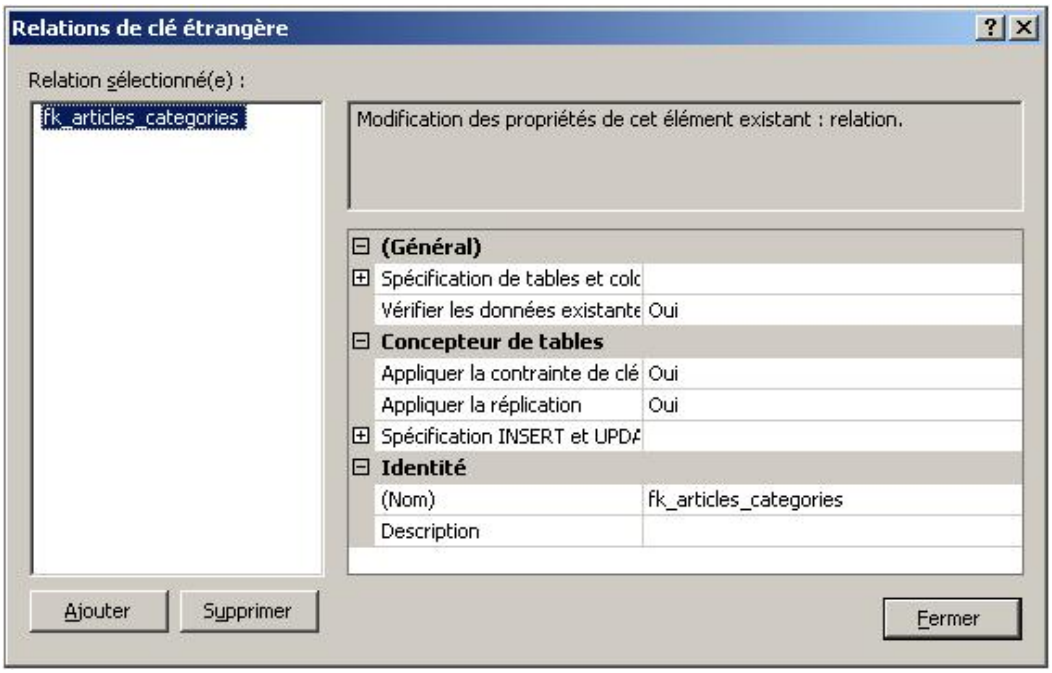
Bien que SQL Server ne comporte pas de limite maximale en ce qui concerne le nombre maximal de contraintes de clé étrangère définies sur chaque table, il est recommandé par Microsoft de ne pas dépasser les 253. Cette limite est à respecter en ce qui concerne le nombre de clés étrangères définies sur 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 nomcolonne]
```

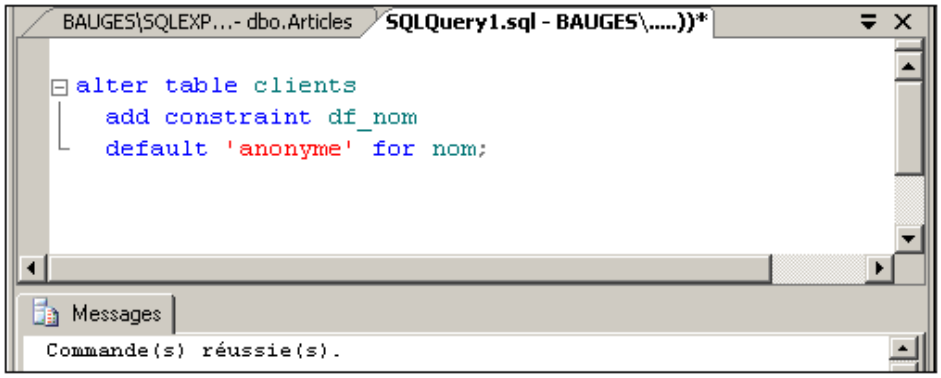
`valeur`

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

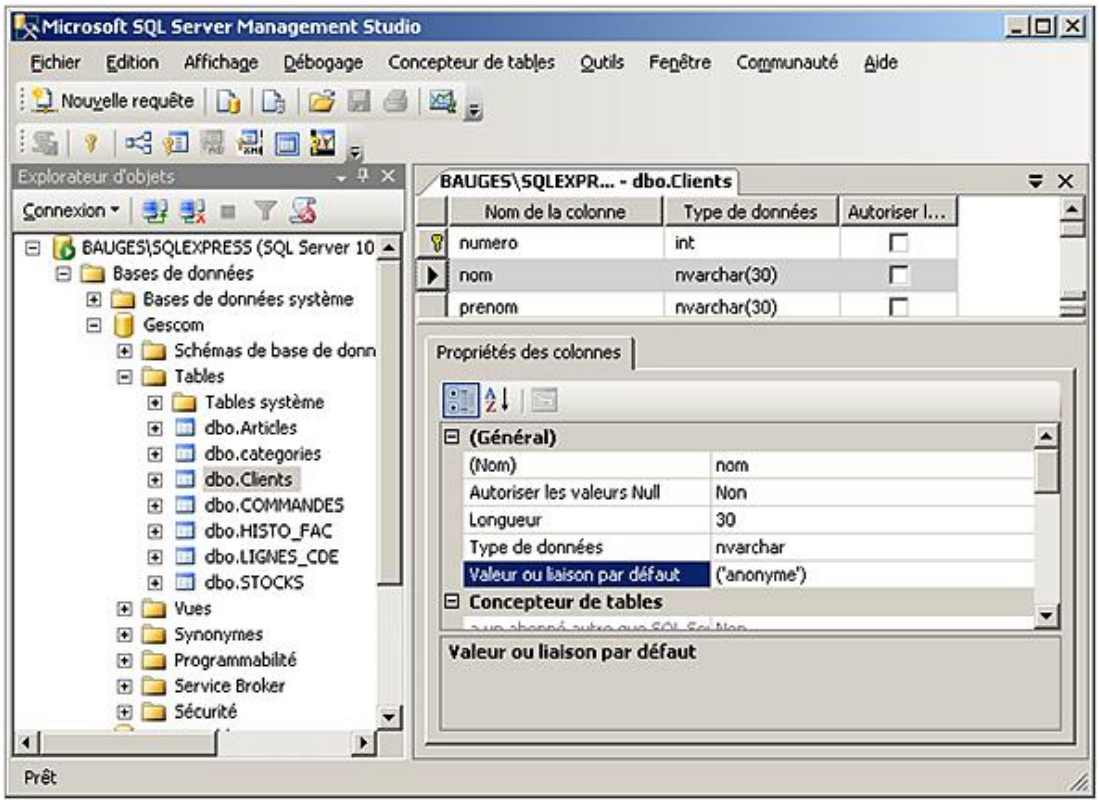
bien la valeur NULL.

*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 Nomcontrainte] CHECK [NOT FOR REPLICATION]
(expression booléenne)
```

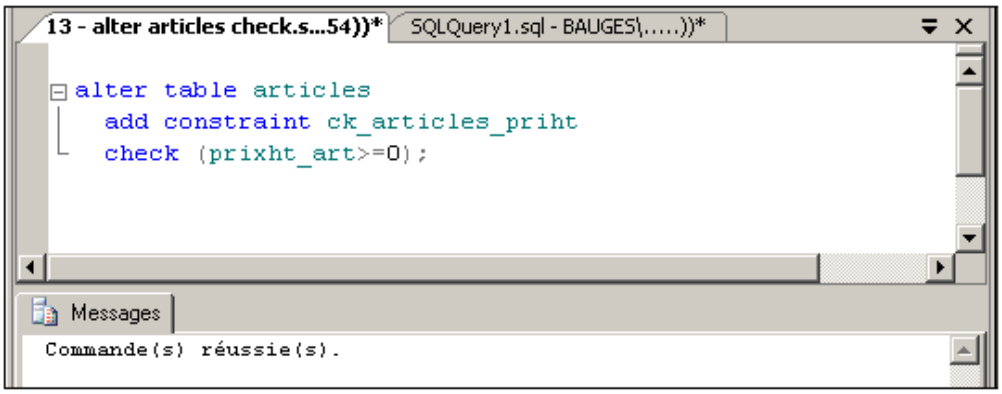
`NOT FOR REPLICATION`

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

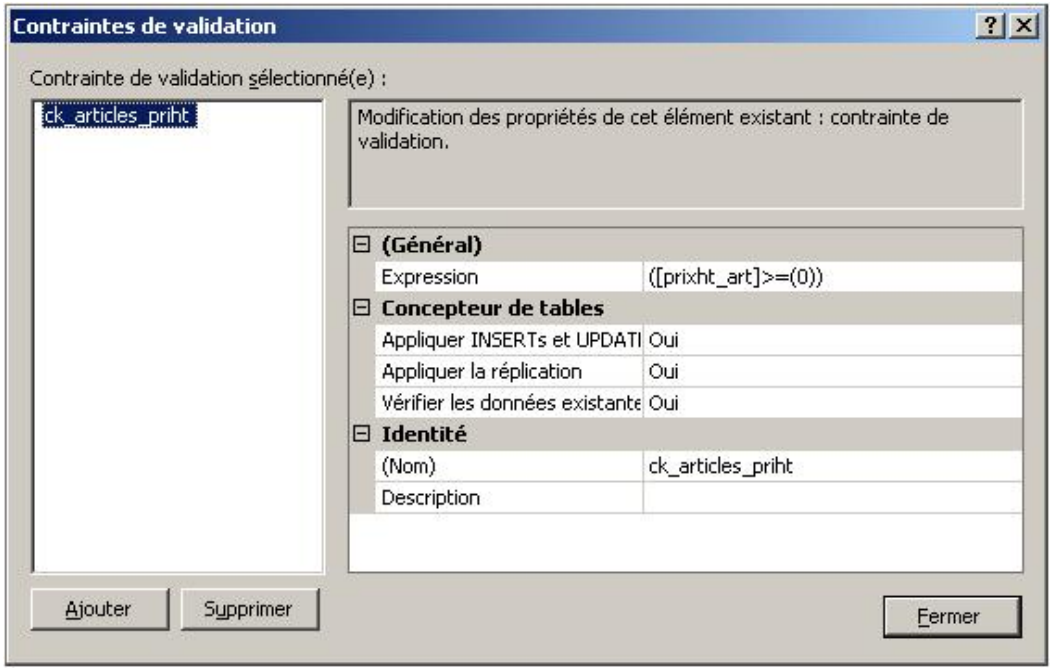
La contrainte CHECK est automatiquement associée à la colonne spécifiée dans l'expression de la condition.

*Exemple*

*Mise en œuvre 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 `prixht<=100/1.196` à la place de `prixht*1.196<=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.

---

Dans l'exemple précédent, tous les articles utilisent une TVA à 19.6%.

---

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 ]
```

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.

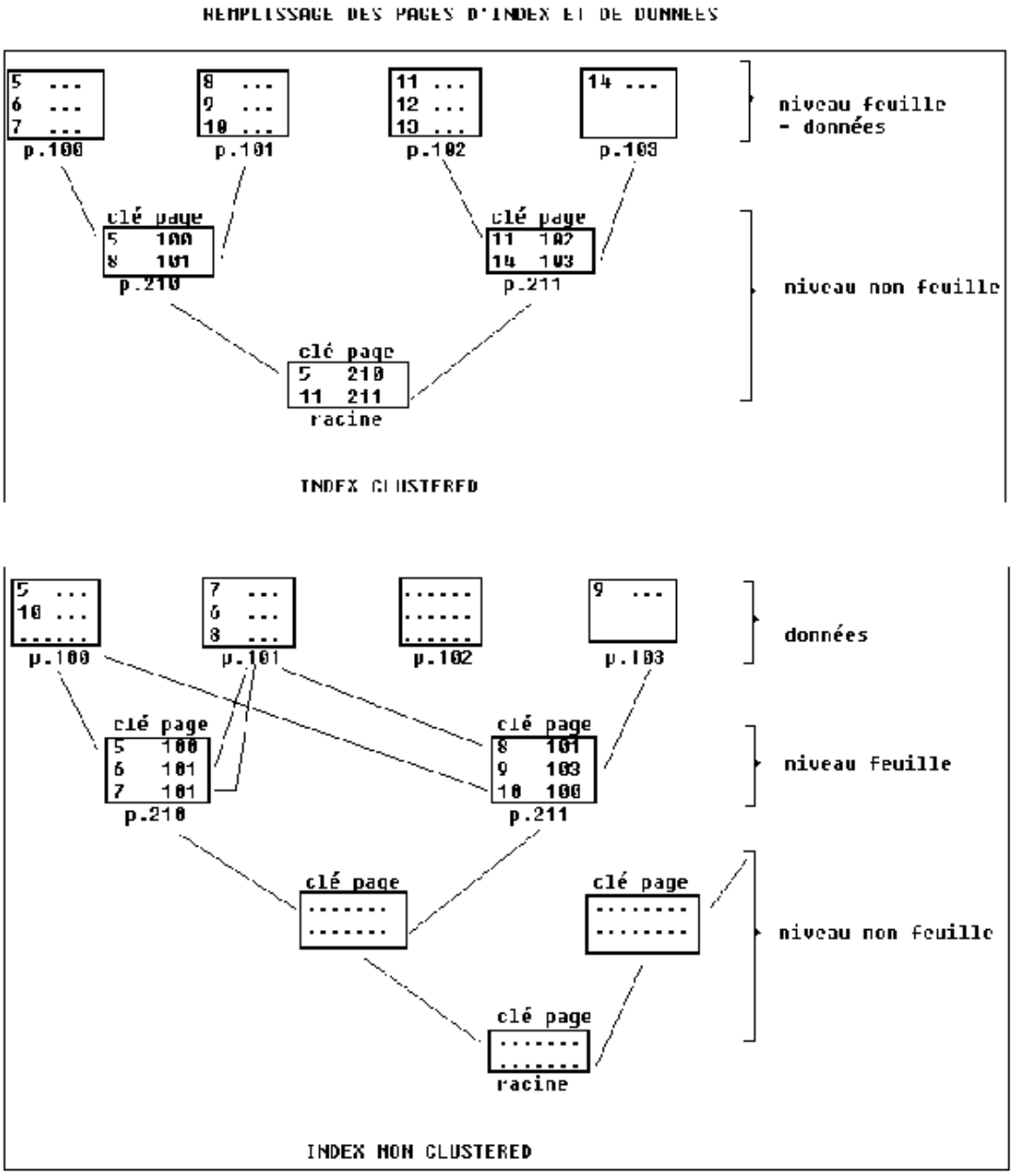
`PAD_INDEX`

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.

`Groupefichier`

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.

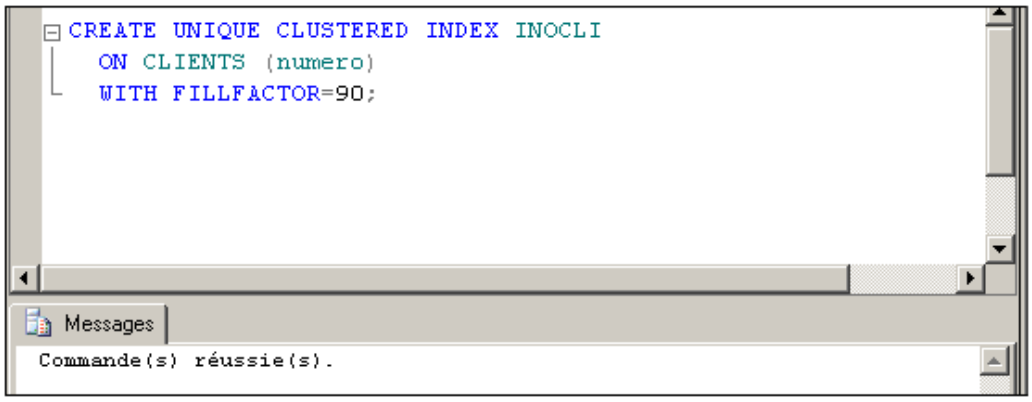
STATISTICS_NORECOMPUTE

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

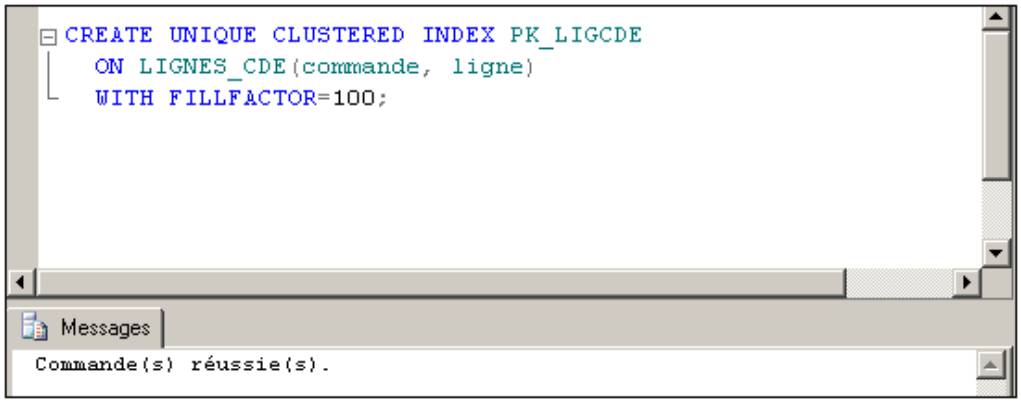
Si le serveur sur lequel s'exécute SQL Server possède plusieurs processeurs, alors la création d'index peut être parallélisée afin de gagner du temps pour la construction d'index sur des tables de grandes dimensions. La mise en place d'un plan d'exécution parallèle pour la construction d'un index prend 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;
```

L'ancienne syntaxe DROP INDEX nomTable.nomIndex est maintenue pour des raisons de compatibilité mais elle n'est pas à utiliser dans le cadre de nouveaux développements.

## 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

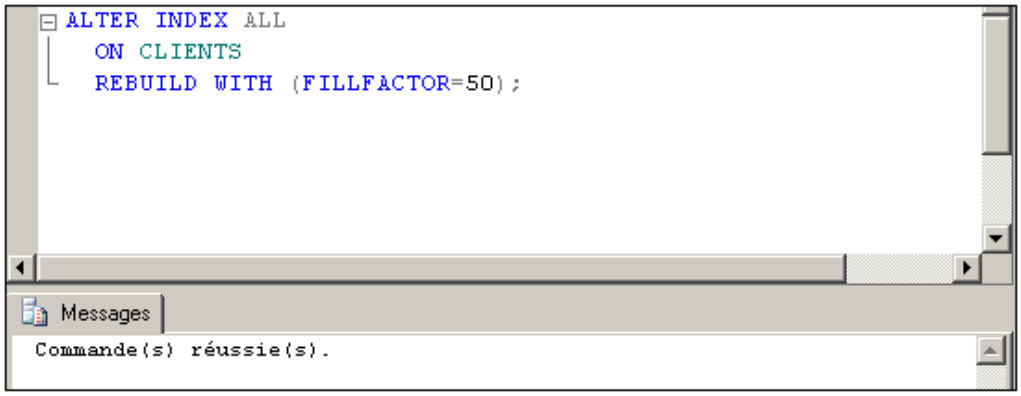
Permet de spécifier le pourcentage de remplissage des pages au niveau feuille de l'index.

PAD_INDEX

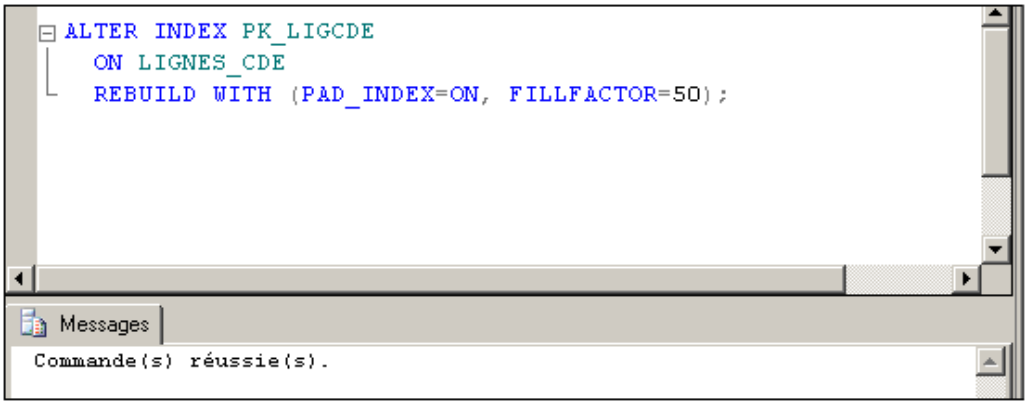
Permet de reporter dans les pages non feuilles de l'index, le même niveau de remplissage que celui précisé par l'intermédiaire de FILLFACTOR.

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.

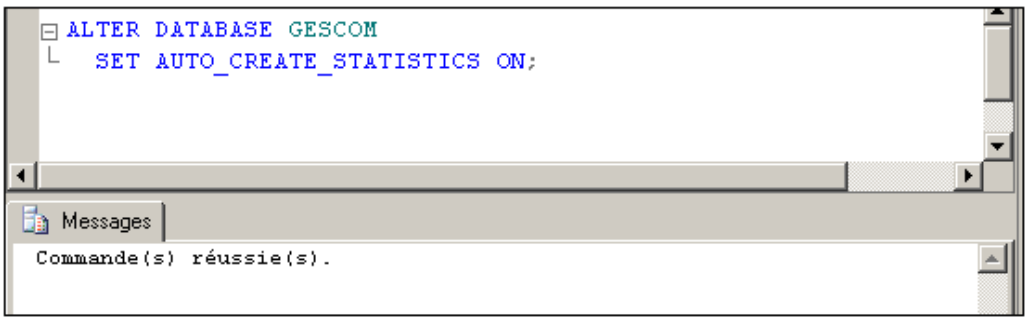
Les statistiques peuvent également être mises à jour de façon automatique. Cette option doit être définie, soit lors de la construction de la base à l'aide de la commande **ALTER DATABASE**, soit en utilisant la procédure stockée **sp_autostats**.

---

La procédure **sp_createstats** permet de définir des statistiques sur tous les index de données utilisateurs de la base en une seule opération.

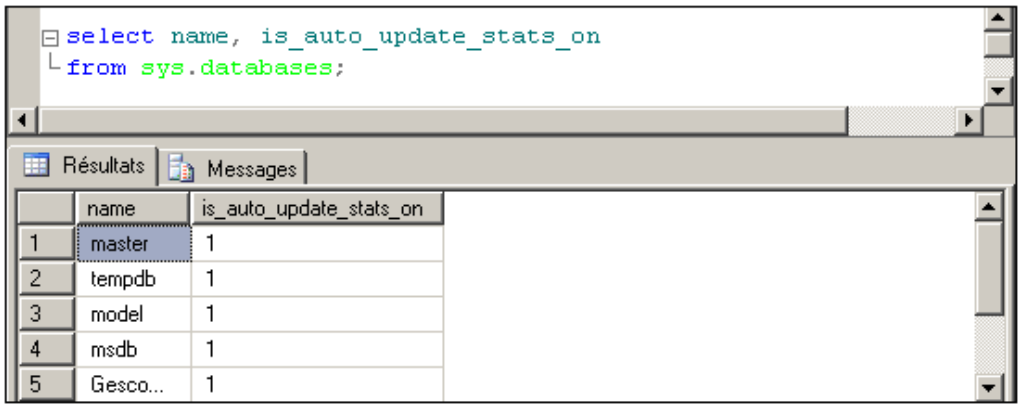
---

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



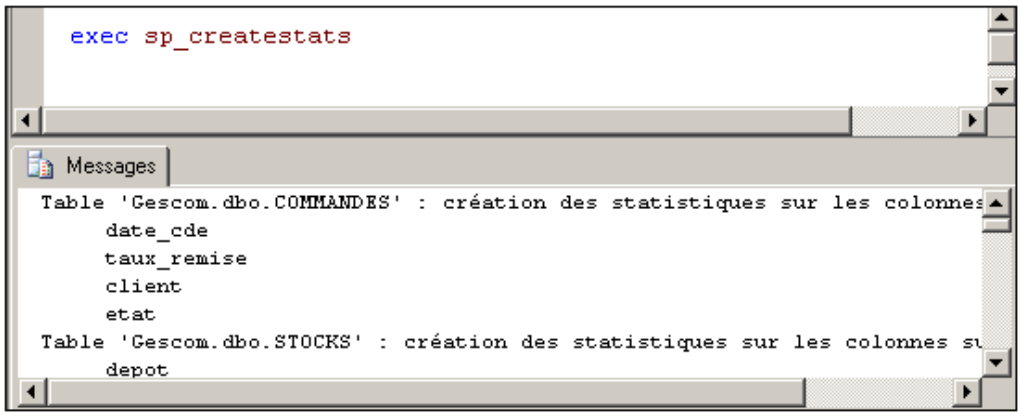

En mode automatique, c'est le moteur qui se charge de calculer les statistiques manquantes, de maintenir à jour les statistiques en fonction des opérations faites sur les données, mais aussi de supprimer les statistiques inutiles.

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**.




| | name | is_auto_update_stats_on |
|---|---|---|
| 1 | master | 1 |
| 2 | tempdb | 1 |
| 3 | model | 1 |
| 4 | msdb | 1 |
| 5 | Gesco... | 1 |

*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**.

---

L'instruction DBCC SHOWCONTIG est maintenue uniquement pour des raisons de compatibilité ascendante. Il est donc conseillé de ne plus l'utiliser.

---

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.

## 6. Surveiller et vérifier les bases et les objets

Après la création et l'utilisation des tables et des index, il est parfois utile de vérifier la cohérence des données et des pages.

L'instruction DBCC le permet.

```
DBCC CHECKDB [(nombase[,NOINDEX])]
```

Elle vérifie pour toutes les tables de la base, la liaison entre pages de données et d'index, les ordres de tri et le pointeur. Des informations sur l'espace disque du journal sont également fournies.

```
DBCC CHECKTABLE (nomtable[,NOINDEX|identification index])
```

Cette instruction effectue un travail similaire à DBCC CHECKDB mais pour une seule table. Si l'identificateur d'index est fourni, seul celui-ci sera vérifié. Si NOINDEX est spécifié, les index ne sont pas vérifiés.

DBCC CHECKFILEGROUP permet quant à lui d'effectuer ces vérifications sur un groupe de fichiers en particulier.

# Généralités

Microsoft Transact SQL est un langage de requêtes amélioré par rapport au SQL dont il reprend les bases. Le SQL (*Structured Query Language*) est le langage standard, créé par IBM dans les années 70, pour la gestion des SGBDR (*Systèmes de Gestion de Bases de Données Relationnelles*).

Trois catégories d'instructions composent ce langage :

- Le langage de Définition de Données (*Data Description Language* - DDL) permettant la création, la modification et la suppression des objets SQL (TABLES, INDEX, VUES, PROCEDURES, etc.).
- Le langage de Manipulation de Données (*Data Manipulation Language* - DML) fournissant les instructions de création, de mise à jour, de suppression et d'extraction des données stockées.
- Le langage de Contrôle d'Accès (*Data Control Language* - DCL) pour la gestion des accès aux données, des transactions et de la configuration des sessions et des bases.

De plus, le Transact SQL prend en compte des fonctionnalités procédurales telles que la gestion des variables, les structures de contrôle de flux, les curseurs, et les lots d'instructions. C'est donc un langage complet qui comporte des instructions, qui manipule des objets SQL, qui admet la programmabilité et qui utilise des expressions.

À l'aide du Transact SQL, il est possible de définir des fonctions et des procédures qui sont directement exécutées sur le serveur de base de données. Ce type de procédure et fonction est particulièrement intéressant lorsque le traitement nécessite de manipuler un volume d'informations important pour produire le résultat. De même, le développement en Transact SQL est parfaitement adapté dans un cadre de partage de fonctionnalités car les procédures et fonctions hébergées par le serveur peuvent être exécutées depuis un environnement client quelconque (.Net, Java…).

À partir de SQL Server 2008, il est possible, mais pas obligatoire, d'utiliser le point-virgule comme marqueur de fin d'instruction.

## 1. Expressions

Dans la plupart des syntaxes Transact SQL, il est possible d'utiliser des expressions ou des combinaisons d'expressions pour gérer des valeurs ou pour tirer parti de la programmabilité du langage.

Les expressions peuvent prendre différentes formes :

### Constantes

*Exemple*

```
Caractère      'AZERTY', 'Ecole Nantaise d''Informatique'
Numérique      10, -15.26, 1.235e-5
Date
constante           date             heure
'801215'            5 Décembre 1980  00:00:00:000
'15/12/1980'        idem             idem
'15-12-80 8:30'     idem             8:30:00:000
'8:30:2'            1er Janvier 1900 08:30:02:000
'15.12.1980 8:30pm' 15 Décembre 1980 20:30:00:000
Binaire         0x05, 0xFF, 0x5aef1b
Nulle           NULL
```

### Noms de colonne

Un nom de colonne pourra être employé comme expression, la valeur de l'expression étant la valeur "stockée" de la colonne.

### Fonctions

On peut utiliser comme expression n'importe quelle fonction, la valeur de l'expression est le résultat retourné par la fonction.

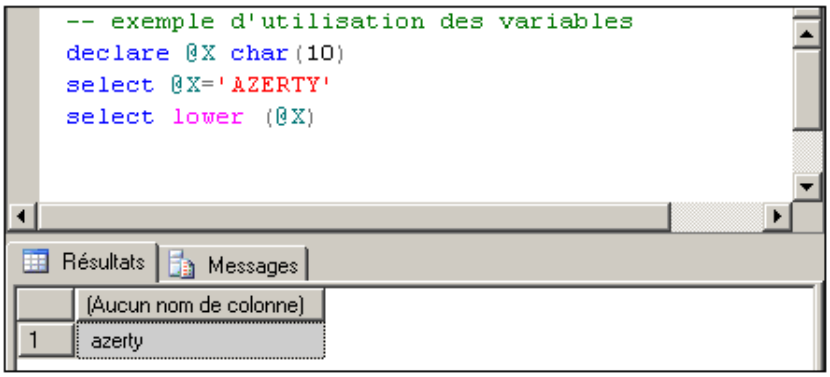
*Exemple*

```
expression                    valeur
SQRT(9)                       3
substring('ABCDEF',2,3)       'BCD'
```

### Variables

Les variables peuvent être employées en tant qu'expression ou dans une expression, sous la forme @nom_de_variable ou @@nom_de_variable. La valeur de l'expression est la valeur de la variable.

*Exemple*

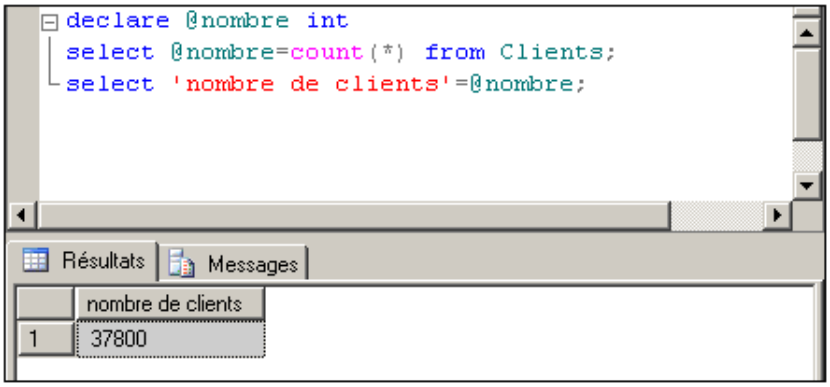



### Sous-requêtes

Une requête SELECT entre parenthèses peut être employée en tant qu'expression ayant pour valeur le résultat de la requête, soit une valeur unique, soit un ensemble de valeurs.

*Exemple*

*Stockage du nombre de clients dans une variable :*




### Expressions booléennes

Elles sont destinées à tester des conditions (IF, WHILE, WHERE, etc.). Ces expressions sont composées de la manière suivante :

```
expression1 opérateur expression2
```

## 2. Opérateurs

Les opérateurs vont permettre de constituer des expressions calculées, des expressions booléennes ou des combinaisons d'expressions.

### Opérateurs arithmétiques

Ils permettent d'effectuer des calculs élémentaires et de renvoyer un résultat.

+ Addition

- Soustraction

* Multiplication

/ Division

% Modulo (reste de la division)

(...) Parenthèses

*Exemple*

```
DECLARE @TTC float
DECLARE @TxTVA float
DECLARE @X int

SELECT @TxTVA=19.6;
SELECT @TTC=(PRIXHT_ART*(1+(@TxTVA/100)))
 FROM ARTICLES
 WHERE REFERENCE_ART='ABO1';

SELECT @X=count(*)
  FROM articles;
IF (@X%2)=0 PRINT 'X est pair'
ELSE PRINT 'X est impair'
```

Messages

```
X est pair
```

Les opérateurs + et - fonctionnent également sur les dates.

## Manipulation de chaînes de caractères

La concaténation permet de constituer une seule chaîne de caractères à partir de plusieurs expressions de type caractère. L'opérateur de concaténation est le signe plus (+).

*Exemple*

```
SELECT 'Monsieur '+nom+' '+prenom as identité
  FROM CLIENTS
 WHERE numero=160163;
```

Résultats | Messages

| | identité |
|---|---|
| 1 | Monsieur Arsenault Ernest |

## Opérateurs de bit

Ils permettent le calcul entre entiers, traduits implicitement en valeurs binaires.

& ET

| OU

^ OU Exclusif

~ NON

### Opérateurs de comparaison

Ils permettent la constitution d'expressions booléennes en comparant des expressions. Ces expressions peuvent être placées entre parenthèses.

`exp1 = exp2`

Égal.

`exp1 > exp2`

Supérieur.

`exp1 >= exp2` ou `exp1!<exp2`

Supérieur ou égal.

`exp1 < exp2`

Inférieur.

`exp1 <= exp2` ou `exp1!>exp2`

Inférieur ou égal.

`exp1 <> exp2` ou `exp1!=exp2`

Différent.

`exp IN (exp1, exp2,...)`

Compare à chaque expression de la liste.

`exp IS NULL`

Test de la valeur NULL. Pour tester si une variable contient la valeur NULL il est indispensable d'utiliser l'opérateur IS NULL.

`exp LIKE 'masque'`

Filtre la chaîne de caractères ou la date suivant le masque spécifié.

Le masque peut être composé de :

_

Un caractère quelconque.

%

n caractères quelconques.

[ab...]

Un caractère dans la liste ab... .

`[a-z]`

Un caractère dans l'intervalle a-z.

`[^ab...]`

Un caractère en dehors de la liste ou de l'intervalle spécifié.

`ab...`

Le caractère lui-même.

Pour utiliser _ , %, [ et ^ en tant que caractères de recherche, il faut les encadrer de [].

*Exemple*

```
masque       Chaînes correspondantes
'G%'         commençant par "G"
'_X%1'       deuxième caractère "X", dernier "1"
'%[1-9]'     se terminant par un chiffre
'[^XW]%'     ne commençant ni par X ni par W
'LE[_]%'     commençant par "LE_"
```

`exp BETWEEN min AND max`

Recherche dans l'intervalle composé des valeurs min et max (bornes incluses).

`EXISTS (sous-requête)`

Renvoie VRAI si la sous-requête a renvoyé au moins une ligne.

### Opérateurs logiques

Ils permettent de combiner des expressions booléennes (expb) en renvoyant une valeur booléenne.

`expb1 OR expb2`

Vrai si une des deux expressions est vraie.

`expb1 AND expb2`

Vrai si les deux expressions sont vraies.

`NOT expb`

Vrai si expb est fausse.

## 3. Fonctions

De nombreuses fonctions sont disponibles pour valoriser des colonnes ou pour effectuer des tests. Des exemples seront donnés avec l'instruction SELECT.

Parmi toutes les fonctions proposées en standard par SQL Server, il est possible de les regrouper par type : rowset, agrégation, ranking, scalaire. Pour ce dernier type de fonctions, il faut en plus les classer par catégories : mathématique, chaîne de caractères, date... car elles sont très nombreuses.

Certaines fonctions, plus particulièrement celles qui travaillent avec des données de type caractère, prennent en compte le classement défini au niveau du serveur.

### Fonctions d'agrégation

Ces fonctions renvoient une valeur unique résultant d'un calcul statistique sur une sélection de lignes.

Les fonctions d'agrégat sont déterministes, c'est-à-dire qu'appliquées à un même ensemble de données, la fonction retournera toujours le même résultat.

À l'exception de la fonction COUNT, les fonctions d'agrégat ne tiennent pas compte des valeurs null.

`COUNT(*)`

Dénombre les lignes sélectionnées.

`COUNT([ALL|DISTINCT] expr)`

Dénombre toutes les expressions non nulles (ALL) ou les expressions non nulles uniques (DISTINCT).

`COUNT_BIG`

Son fonctionnement est identique à celui de la fonction COUNT mais le résultat retourné est au format bigint à la place de int pour la fonction COUNT.

`SUM([ALL|DISTINCT] exprn)`

Somme de toutes les expressions non nulles (ALL) ou les expressions non nulles uniques (DISTINCT).

`AVG([ALL|DISTINCT] exprn)`

Moyenne de toutes les expressions non nulles (ALL) ou les expressions non nulles uniques (DISTINCT).

`MIN(exp)` ou `MAX(exp)`

Valeur minimum ou maximum de toutes les expressions.

`STDEV ([ALL|DISTINCT] exprn)`

Écart type de toutes les valeurs de l'expression donnée.

`STDEVP ([ALL|DISTINCT] exprn)`

Écart type de la population pour toutes les valeurs de l'expression donnée.

`VAR ([ALL|DISTINCT] exprn)`

Variance de toutes les valeurs de l'expression donnée.

`VARP ([ALL|DISTINCT] exprn)`

Variance de la population pour toutes les valeurs de l'expression donnée.

`GROUPING`

S'utilise conjointement à ROLLUP et CUBE. Indique la valeur 1 lorsque la ligne est générée par une instruction ROLLUP ou CUBE, sinon elle indique la valeur 0.

`CHECKSUM (*|exp[,...])`

Permet de calculer un code de contrôle par rapport à une ligne de la table ou par rapport à une liste d'expressions, plusieurs colonnes par exemple. Cette fonction permet la production d'un code de hachage.

`CHECKSUM_AGG([ALL|DISTINCT]exp)`

Permet le calcul d'une valeur de hachage par rapport à un groupe de données. Ce code de contrôle permet de savoir rapidement si des modifications ont eu lieu sur un groupe de données, car la valeur produite par cette fonction n'est plus la même.

## Fonctions mathématiques

Ces fonctions renvoient une valeur résultant de calculs mathématiques classiques (algèbre, trigonométrie, logarithmes, etc.).

`ABS(expn)`

Valeur absolue de `expn`.

`CEILING(expn)`

Plus petit entier supérieur ou égal à `expn`.

`FLOOR(expn)`

Plus grand entier inférieur ou égal à `expn`.

`SIGN(expn)`

Renvoie 1 si `expn` est positive, -1 si elle est négative et 0 si elle est égale à zéro.

`SQRT(expn)`

Racine carrée de `expn`.

`POWER(expn,n)`

`expn` à la puissance `n`.

`SQUARE(expn)`

Calcul du carré de `expn`.

## Fonctions trigonométriques

`PI()`

Valeur du nombre PI.

`DEGREES(expn)`

Conversion en degrés de `expn` en radians.

`RADIANS(expn)`

Conversion en radians de `expn` en degrés.

`SIN(expn), TAN(expn), COS(expn), COT(expn)`

Sinus, tangente, cosinus et cotangente de l'angle `expn` en radians.

`ACOS(expn), ASIN(expn), ATAN(expn)`

Arc cosinus, arc sinus et arc tangente de `expn`.

`ATN2(expn1,expn2)`

Angle dont la tangente se trouve dans l'intervalle `expn1`, `expn2`.

## Fonctions logarithmiques

`EXP(expn)`

Exponentielle de expn.

`LOG(expn)`

Logarithme népérien de expn.

`LOG10(expn)`

Logarithme base 10 de expn.

### Fonctions diverses

`RAND([expn])`

Nombre aléatoire compris entre 0 et 1. Expn représente la valeur de départ.

`ROUND(expn,n[,f])`

Arrondit expn à la précision n. Si n est positif, n représente le nombre de décimales. Si n est égal à zéro, arrondit à l'entier le plus proche. Si n est négatif, arrondit à la dizaine la plus proche (-1), à la centaine (-2), etc., ou renvoie 0 si n est supérieur au nombre de chiffres entiers de expn. Si f est précisé, son rôle est de tronquer expn. Les valeurs pouvant être prises par f sont interprétées comme pour n. La valeur de f ne sera prise en compte que si n est égal à 0.

*Exemple*

```
expn          n         f          résultat
1.256         2                    1.260
1.256         4                    1.256
1.256         0                    1.000
11.25        -1                   10
11.25        -2                     .00
11.25        -3                     .00
150.75        0                  151
150.75        0        1         150
```

### Fonctions date

Les fonctions date manipulent des expressions de type DATETIME, et utilisent des formats représentant la partie de la date à gérer.

Ces formats sont :

| Format | Abréviation | Signification |
|---|---|---|
| year | yy, yyyy | Année (de 1753 à 9999) |
| quarter | qq, q | Trimestre (1 à 4) |
| month | mm, m | Mois (1 à 12) |
| day of year | dy, y | Jour dans l'année (1 à 366) |
| day | dd, d | Jour dans le mois (1 à 31) |
| weekday | dw, ww | Jour dans la semaine (1 Lundi à 7 Dimanche) |
| hour | hh | Heure (0 à 23) |
| minute | mi, n | Minute (0 à 59) |

| | | |
|---|---|---|
| seconds | ss, s | Seconde (0 à 59) |
| millisecond | ms | Milliseconde (0 à 999) |

`GETDATE()`

Date et heure système.

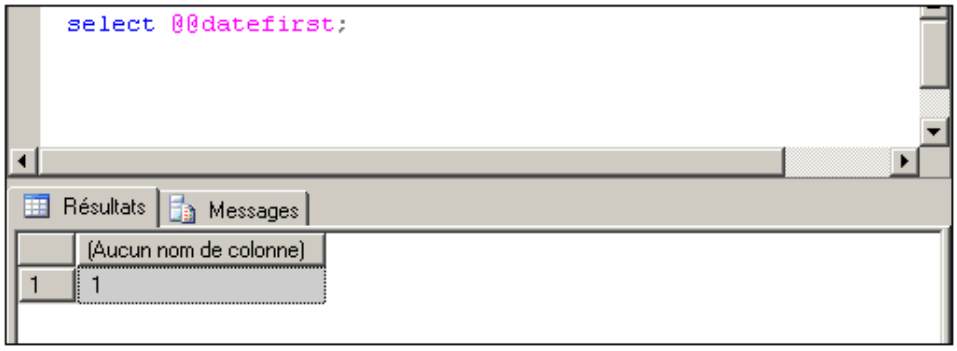
`DATENAME(format,expd)`

Renvoie la partie date sous forme de texte.

`DATEPART(format,expd)`

Renvoie la valeur de la partie date selon le format.

Il est possible de configurer le premier jour de la semaine par l'intermédiaire de la fonction SET DATEFIRST (numero_jour). Les jours sont numérotés de 1 pour le lundi à 7 pour le dimanche. Il est possible de connaître la configuration actuelle en interrogeant la fonction @@datefirst.




`DATEDIFF(format,expd1,expd2)`

Différence selon le format entre les deux dates.

`DATEADD(format,n,expd)`

Ajoute `n format` à la date `expd`.

`DAY(expd)`

Retourne le numéro du jour dans le mois.

`MONTH(expd)`

Retourne le numéro du mois.

`YEAR(expd)`

Retourne l'année.

---

Pour les trois fonctions précédentes, si l'expression passée en paramètre est la valeur 0, alors SQL Server effectue ses calculs sur le 1er janvier 1900.

---

`SWITCHOFFSET (datetimeoffset, zoneHoraire)`

Convertit la donnée de type `DATETIMEOFFSET` passée en paramètre vers la zone horaire passée en second paramètre.

`SYSDATETIME`

Retourne la date et heure courante du serveur au format datetime2. Le décalage par rapport au temps GMT (OFFSET) n'est pas inclus. Cette fonction offre plus de précision que la valeur retournée par getdate ou bien getutcdate.

`SYSDATETIMEOFFSET`

Fonctionnement similaire à `SYSDATETIME` mais la donnée de type date et heure retournée est au format datetimeoffset et inclut donc le décalage par rapport à l'heure GMT.

### Fonctions chaîne de caractères

`ASCII(expc)`

Valeur du code ASCII du premier caractère de `expc`.

`UNICODE(expc)`

Valeur numérique correspondant au code caractère Unicode de `expc`.

`CHAR(expn)`

Caractère correspondant au code ASCII `expn`.

`NCHAR(expn)`

Caractère unicode correspondant au code numérique `expn`.

`LTRIM(expc), RTRIM(expc)`

Supprime les espaces non significatifs à droite (RTRIM) ou à gauche (LTRIM) de `expc`.

`STR(expn,[lg[,nbd]])`

Convertit le nombre `expn` en chaîne de longueur totale `lg` dont `nbd` caractères à droite de la marque décimale.

`SPACE(n)`

Renvoie `n` espaces.

`REPLICATE(expc,n)`

Renvoie `n` fois `expc`.

`CHARINDEX('masque',expc) PATINDEX ('%masque%',expc)`

Renvoie la position de départ de la première expression 'masque' dans `expc`. `PATINDEX` permet d'utiliser des caractères génériques (voir LIKE) et de travailler avec les types `text`, `char` et `varchar`.

`LOWER(expc), UPPER(expc)`

Changement de casse. Convertit `expc` en minuscule ou en majuscule.

`REVERSE(expc)`

Retourne `expc` à l'envers (lue de droite à gauche).

`RIGHT(expc,n)`

Renvoie les `n` caractères les plus à droite de `expc`.

`LEFT(expc,n)`

Renvoie les n caractères les plus à gauche de expc.

```
SUBSTRING(expc,dp,lg)
```

Renvoie lg caractères de expc à partir de dp.

```
STUFF(expc1,dp,lg,expc2)
```

Supprime lg caractères de expc1 à partir de dp, puis insère expc2 à la position dp.

```
SOUNDEX(expc)
```

Renvoie le code phonétique de expc. Ce code est composé de la première lettre de expc et de trois chiffres.

```
DIFFERENCE(expc1,expc2)
```

Compare les SOUNDEX des deux expressions. Renvoie une valeur de 1 à 4, 4 signifiant que les deux expressions offrent la plus grande similitude.

```
LEN(expc)
```

Retourne le nombre de caractères de expc.

```
QUOTENAME(expc[,delimiteur])
```

Permet de transformer expc en identifiant valide pour SQL Server.

```
REPLACE(expc1, expc2, expc3)
```

Permet de remplacer dans expc1 toutes les occurrences de expc2 par expc3.

### Fonctions système

```
COALESCE (exp1,exp2,...)
```

Renvoie la première exp non NULL.

```
COL_LENGTH (’nom_table’,’nom_colonne’)
```

Longueur de la colonne.

```
COL_NAME (id_table,id_col)
```

Nom de la colonne dont le numéro d’identification est id_col dans la table identifiée par id_table.

```
DATALENGTH (exp)
```

Longueur en octets de l’expression.

```
DB_ID([’nom_base’])
```

Numéro d’identification de la base de données.

```
DB_NAME ([id_base])
```

Nom de la base identifiée par id_base.

```
GETANSINULL ([ ’nom_base’])
```

Renvoie 1 si l’option "ANSI NULL DEFAULT" est positionnée pour la base.

`HOST_ID()`

Numéro d'identification de la station de travail.

`HOST_NAME()`

Nom de la station de travail.

`IDENT_INCR ('nom_table')`

Valeur de l'incrémentation définie pour la colonne IDENTITY de la table ou de la vue portant sur une table ayant une colonne IDENTITY.

`IDENT_SEED ('nom_table')`

Valeur initiale définie pour la colonne IDENTITY de la table, ou de la vue portant sur une table ayant une colonne IDENTITY.

`IDENT_CURRENT ('nom_table')`

Retourne la dernière valeur de type identité utilisée par cette table.

`INDEX_COL ('nom_table',id_index,id_cle)`

Nom de la colonne indexée correspondant à l'index.

`ISDATE(exp)`

Renvoie 1 si l'expression de type varchar a un format de date valide.

`ISNULL(exp,valeur)`

Renvoie valeur si `exp` est NULL.

`ISNUMERIC(exp)`

Renvoie 1 si l'expression de type varchar a un format numérique valide.

`NULLIF(exp1,exp2)`

Renvoie NULL si `exp1 = exp2`.

`OBJECT_ID('nom')`

Numéro d'identification de l'objet 'nom'.

`OBJECT_NAME(id)`

Nom de l'objet identifié par `id`.

`STATS_DATE(id_table,id_index)`

Date de dernière mise à jour de l'index.

`SUSER_SID(['nom_acces'])`

Numéro d'identification correspondant au nom d'accès.

`SUSER_SNAME([id])`

Nom d'accès identifié par `id`.

`USER_NAME([id])`

Nom de l'utilisateur identifié par `id`. À n'utiliser qu'avec la contrainte DEFAULT (fonctions niladiques).

`CURRENT_TIMESTAMP`

Date et heure système, équivalent à GETDATE().

`SYSTEM_USER`

Nom d'accès.

`CURRENT_USER, USER, SESSION_USER`

Nom de l'utilisateur de la session.

`OBJECT_PROPERTY(id,propriété)`

Permet de retrouver les propriétés d'un objet de la base.

`ROW_NUMBER`

Permet de connaître le numéro séquentiel d'une ligne issue d'une partition depuis un jeu de résultats. Cette numérotation commence à 1 pour chaque première ligne de chaque partition.

`RANK`

Permet de connaître le rang d'une ligne issue d'une partition depuis un jeu de résultats. Le rang d'une ligne est supérieur d'une unité au rang de la ligne issue de la même partition.

`DENSE_RANK`

Fonctionne comme `RANK` mais ne s'applique qu'aux lignes présentes dans le jeu de résultats.

`HAS_DBACCESS('nomBase')`

Permet de savoir si, avec le contexte de sécurité actuel, il est possible d'accéder à la base de données passée en paramètre (retour=1) ou si ce n'est pas possible (retour=0).

`HAS_PERMS_BY_NAME`

Permet de savoir par programmation si l'on dispose d'un privilège ou non. Ce type de fonction peut s'avérer intéressant dans le cadre d'un changement de contexte.

`KILL`

Cette fonction, bien connue des utilisateurs des systèmes Unix/Linux, permet de mettre fin à une session utilisateur. Pour mettre fin à une connexion, il est nécessaire de passer en paramètre l'identifiant de session (sessionID ou SPID) ou bien l'identifiant du lot actuellement en cours d'exécution (UOW - *Unit Of Work*). Le UOW peut être connu en interrogeant la colonne request_owner_guid de la vue sys.dm_tran_locks. L'identifiant de session peut quant à lui être obtenu en interrogeant le contenu de la variable @@SPID, en exécutant la procédure sp_who ou bien encore en interrogeant la colonne session_id de vues telles que sys.dm_tran_locks ou sys_dm_exec_sessions. Si l'annulation porte sur une transaction de grande envergure alors l'opération peut être relativement longue.

`NEWID()`

Permet de générer une valeur de type UniqueIdentifier.

`NEWSEQUENTIALID()`

Cette fonction, destinée à être utilisée uniquement pour générer une valeur par défaut, permet de générer la prochaine valeur de type UniqueIdentifier. Comme la prochaine valeur de type UniqueIdentifier est prédictible, cette fonction ne doit pas être mise en place dans les structures nécessitant un haut niveau de sécurité.

`PARSENAME('nomObjet',partieAExtraire)`

Permet d'extraire à partir d'un nom complet d'objet le nom de l'objet (*partieAExtraire=1*), le nom du schéma (*partieAExtraire=2*), le nom de la base de données (*partieAExtraire=3*) ou bien le nom du serveur (*partieAExtraire=4*).

`PUBLISHINGSERVERNAME`

Permet de savoir qui est à l'origine d'une publication.

`STUFF (chaine1, dp, lg, chaine2)`

Permet de supprimer `lg` caractère de la chaîne `chaine1` à partir de la position `dp`, puis d'y insérer `chaine2`.

### Fonctions conversion de type

`CAST(expression AS type_donnée)`

Permet de convertir une valeur dans le type spécifié.

`CONVERT(type_donnée,exp[,style])`

Conversion de l'expression dans le type spécifié. Un style peut être utilisé dans les conversions date ou heure.

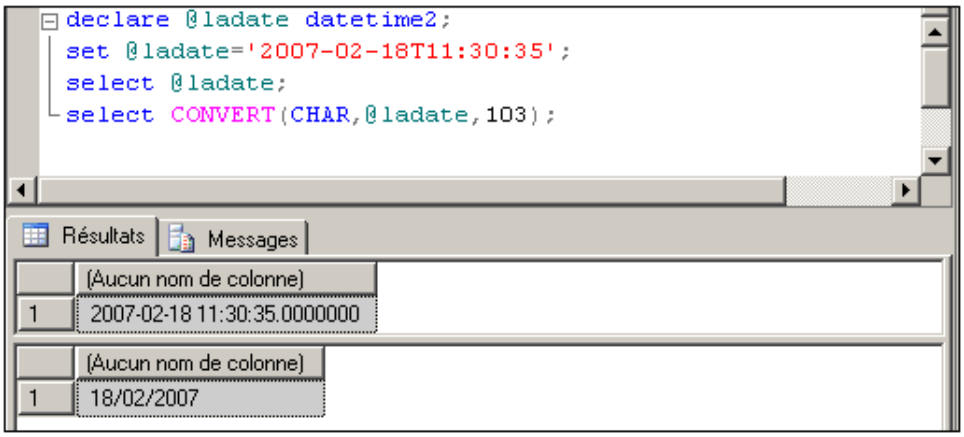
| Sans le siècle | Avec le siècle | Standard | Sortie |
|---|---|---|---|
| - | 0 ou 100 | défaut | (mois/mmm)jj aaaa hh:mi AM (ou PM) |
| 1 | 101 | USA | mm/jj/aa |
| 2 | 102 | ANSI | aa.mm.jj |
| 3 | 103 | anglais/français | jj/mm/aa |
| 4 | 104 | allemand | jj.mm.aa |
| 5 | 105 | italien | jj-mm-aa |
| 6 | 106 | - | jj mmm aa |
| 7 | 107 | - | mmm jj,a |
| 8 | 108 | - | hh:mi:ss |
| - | 9 ou 109 | défaut | (mois/mmm)jj aaaa hh:mi:ss:mmm AM |
| 10 | 110 | USA | mm-jj-aa |
| 11 | 111 | Japon | aa/mm/jj |
| 12 | 112 | ISO | aammjj |
| - | 13 ou 113 | européen | jj(mois/mmm)aaaa hh:mi:ss:mmm(24h) |
| 14 | 114 | - | hh:mi:ss:mmm (24h) |
| - | 20 ou 120 | ODBC canonique | yyyy-mm-jj hh:mi:ss(24h) |
| - | 21 ou 121 | ODBC canonique | yyyy-mm-jj hh:mi:ss.mmm(24h) |
| - | 126 | ISO8601 | yyyy-mm-jjThh:mi:ss.mmm |

| | | | |
|---|---|---|---|
| - | 130 | Hijri | dd mois yyyy hh:mi:ss:mmmAM |
| - | 131 | Hijri | dd/mm/yy hh:mi:ss:mmmAM |

Dans le cas où l'année est saisie sur deux caractères, alors SQL Server utilise 2049 comme année charnière, c'est-à-dire que pour toute valeur comprise entre 0 et 49 SQL Server considère qu'il s'agit d'une année du XXI$^{ème}$ siècle tandis que pour toute valeur comprise entre 50 et 99 SQL Server considère qu'il s'agit d'une année du XX$^{ème}$ siècle. Ainsi 08 correspond à 2008 tandis que 89 correspond à 1989. Bien entendu, cette date charnière est un paramètre de SQL Server et peut être configurée en conséquence.

## Travailler avec les dates

Pour travailler de façon simple avec les dates et éviter toute erreur de conversion et tout problème de formatage, l'opération la plus simple consiste à s'appuyer sur le format ISO 8601. Comme son nom l'indique ce format est normalisé et permet la transformation aisée d'une chaîne de caractères en donnée de type date/heure. Dans le format ISO 8601, la date est exprimée ainsi :yyyy-mm-ddThh:mm:ss[.mmm]. Toutes les informations année, mois, jour, heure (sur 24 H) minutes et secondes doivent être valorisées. Seules les informations relatives aux millièmes de seconde sont optionnelles. Dans ce format, les séparateurs sont également obligatoires et ne peuvent pas être modifiés.



# Le SQL-DML

Le langage de manipulation de données (*SQL-Data Manipulation Language*), est composé des instructions permettant la gestion, la visualisation et l'extraction des lignes des tables et des vues.

## 1. Création de lignes

La création de lignes dans une table, ou dans une vue selon certaines conditions, se fait par la commande INSERT.

Les contraintes seront contrôlées et les triggers seront déclenchés lors de l'exécution de la commande.

La forme "INSERT ..... VALUES ....." crée une seule ligne, alors que "INSERT ..... SELECT ....." permet de créer éventuellement plusieurs lignes.

**Syntaxe**

```
INSERT [INTO] nom_objet [(col,...)]{DEFAULT VALUES|VALUES
(val,...)|requête|EXECUTE procedure}
```

`nom_objet`

Nom valide de table ou de vue.

`(col,...)`

Liste des colonnes à valoriser. Les colonnes non citées prendront la valeur NULL. Si la liste est omise, toutes les colonnes devront être associées à une valeur.

`DEFAULT VALUES`

Toutes les colonnes citées prendront leur valeur par défaut ou NULL si elles n'en ont pas.

`(val,...)`

Liste de valeurs composée d'expressions constantes, des mots clés NULL ou DEFAULT, ou de variables. Il doit y avoir autant de valeurs que de colonnes à valoriser, du même type, et dans le même ordre.

`requête`

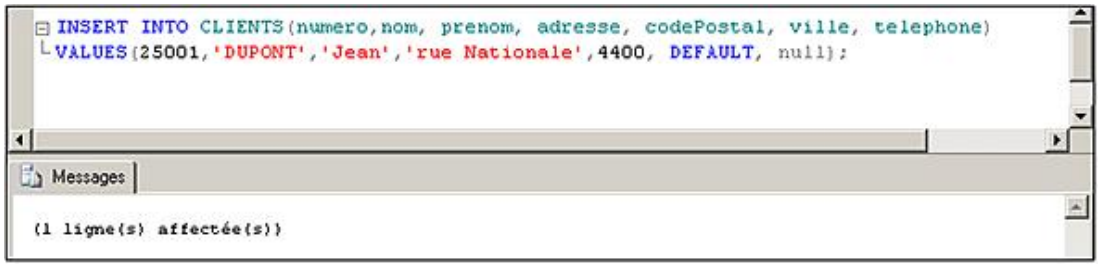
Instruction SELECT renvoyant autant de valeurs, dans le même ordre et du même type que les colonnes à valoriser. Cette forme de syntaxe permet d'insérer plusieurs lignes en une seule opération.

`procedure`

Nom d'une procédure stockée locale ou distante. Seules les valeurs des colonnes renvoyées par un ordre SELECT contenu dans la procédure valoriseront les colonnes concernées par l'INSERT.

*Exemples*

*Création d'un Client avec la valorisation de toutes les colonnes, valeur par défaut pour la ville, NULL pour le téléphone :*




*Création d'un article (prix inconnu) :*

```
INSERT INTO ARTICLES(reference_art, designation_art, code_cat)
VALUES ('SS2K8P','SQL Server 2008',53);
```

```
Messages

(1 ligne(s) affectée(s))
```

*Création d'articles en stock dans le dépôt P2 à partir de la table Articles :*

```
INSERT INTO STOCKS(article, depot, quantite)
SELECT REFERENCE_ART,'P2',0 FROM ARTICLES;
```

```
Messages

(9422 ligne(s) affectée(s))
```

*Exécution d'une procédure ramenant les numéros et noms des tables d'un utilisateur identifié par son numéro :*

```
exec noms_tables 1
```

Résultats | Messages

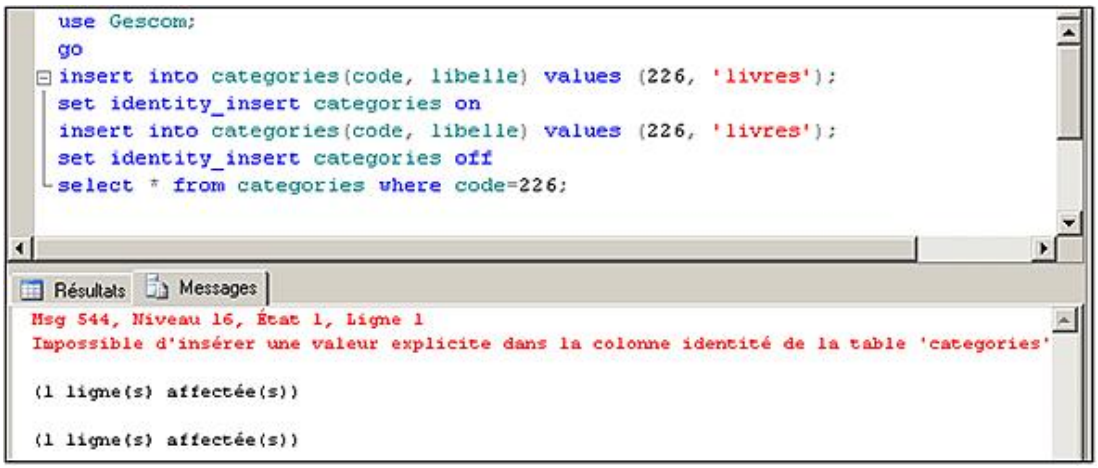
| | object_id | name |
|---|---|---|
| 1 | 53575229 | COMMANDES |
| 2 | 117575457 | STOCKS |
| 3 | 149575571 | HISTO_FAC |
| 4 | 197575742 | categories |
| 5 | 229575856 | LIGNES_CDE |
| 6 | 2105058... | Articles |
| 7 | 2121058... | Clients |

*Application de cette procédure pour valoriser une table temporaire avec les informations sur les tables dont le propriétaire est "dbo" :*

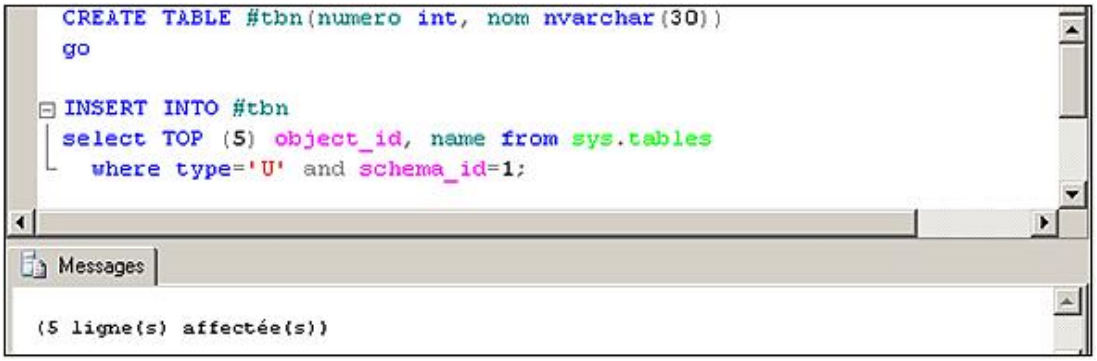
```
CREATE TABLE #tbn(numero int, nom nvarchar(30))
go
DECLARE @IDUtilisateur int
SELECT @IDUtilisateur=USER_ID('dbo')
INSERT INTO #tbn execute dbo.noms_tables @IDUtilisateur;
```

```
Messages

(7 ligne(s) affectée(s))
```

*Insertion de lignes en forçant la valeur pour une colonne de type Identity :*




Lorsque l'instruction INSERT est associée à une requête SELECT, il est possible de limiter le nombre de lignes insérées en utilisant la clause TOP.




L'instruction INSERT permet d'insérer plusieurs lignes de données de façon simultanée lorsqu'elle est associée à une clause SELECT, mais il est également possible d'ajouter plusieurs lignes en précisant directement les valeurs des données à insérer. Pour cela, les données relatives à chaque ligne à insérer sont spécifiées entre parenthèses derrière la clause VALUES et chaque ensemble de valeurs est séparé par une virgule.

**Syntaxe**

```
INSERT INTO nom_table[(col, ...)]
VALUES((valeurLigne1, ...),(valeurLigne2,...), ...);
```

`nom_table`

Nom de la table concernée par l'opération d'ajout de lignes.

`(col, …)`

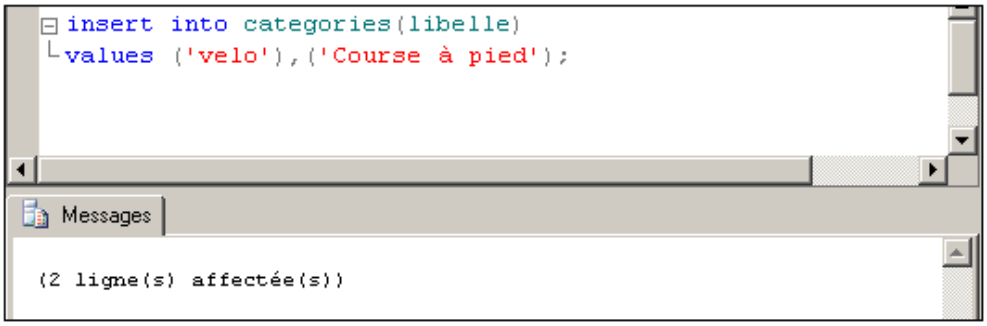
Nom des différentes colonnes de la table pour lesquelles des valeurs sont fournies. L'ordre des colonnes défini à cet endroit définit également l'ordre dans lequel les valeurs doivent être fournies.

`(valeurLigne1, …)`

Valeurs des données à ajouter dans la table, chaque ensemble de valeurs doit fournir une donnée pour chaque colonne spécifiée dans la liste des colonnes.

*Exemple*

*Dans l'exemple suivant deux valeurs sont insérées dans la table catégories à l'aide d'une seule instruction INSERT :*




## 2. Modification de lignes

La modification des valeurs des colonnes de lignes existantes s'effectue par l'instruction UPDATE. Cette instruction peut mettre à jour plusieurs colonnes de plusieurs lignes d'une table à partir d'expressions ou à partir de valeurs d'autres tables.

**Syntaxe**

```
UPDATE nom_objet SET col=exp[,...]
[FROM nom_objet[,...]][WHERE cond]
```

`nom_objet`

Nom de table ou de vue.

`col`

Nom de la colonne à mettre à jour.

`exp`

N'importe quel type d'expression renvoyant une seule valeur de même type que la colonne. Les mots clés DEFAULT et NULL sont autorisés.

`FROM nom_objet`

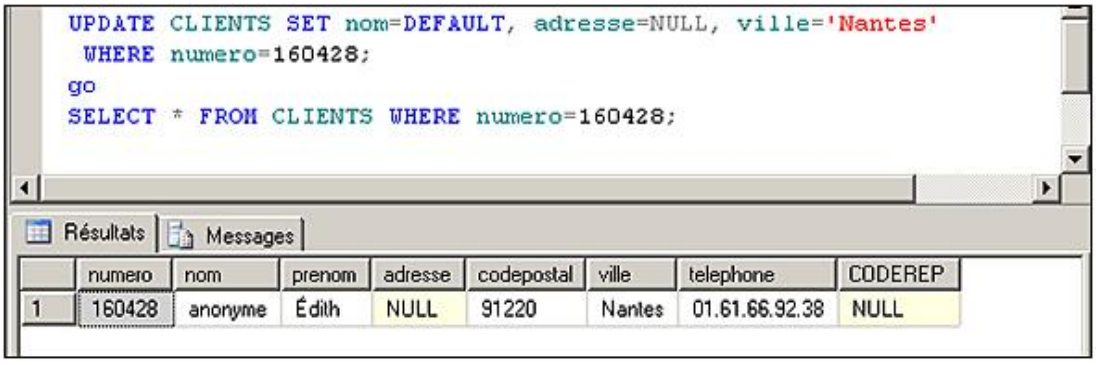
Permet de mettre à jour chaque ligne à partir des données d'autres tables ou vues.

`cond`

Expression booléenne permettant de limiter les lignes à mettre à jour.

*Exemples*

*Modification des données NOM, ADRESSE et VILLE du Client 160428 :*




*Augmentation de 1 % des prix de tous les articles de moins de 10 € :*

```
UPDATE ARTICLES SET prixht_art=prixht_art*1.01
 WHERE prixht_art<10;
```

Messages

```
(693 ligne(s) affectée(s))
```

*Mise à jour de la quantité en stock de l'article portant la référence 000428 dans le Dépôt N1 par rapport à toutes les lignes de commande concernant cet article :*

```
UPDATE STOCKS
   SET quantite=quantite-(SELECT SUM(quantite)
                          FROM LIGNES_CDE l
                         WHERE l.article=stocks.article)
 WHERE stocks.article='000428'
   AND stocks.depot='N1';
```

Messages

```
(0 ligne(s) affectée(s))
```

## 3. Suppression de lignes

L'instruction DELETE permet de supprimer une ou plusieurs lignes d'une table ou vue en fonction d'une condition ou des données d'une autre table.

**Syntaxe**

```
DELETE [FROM] nom_objet [FROM nom_objet [,...]][WHERE cond]
```

`nom_objet`

Nom de table ou de vue.

`FROM nom_objet`

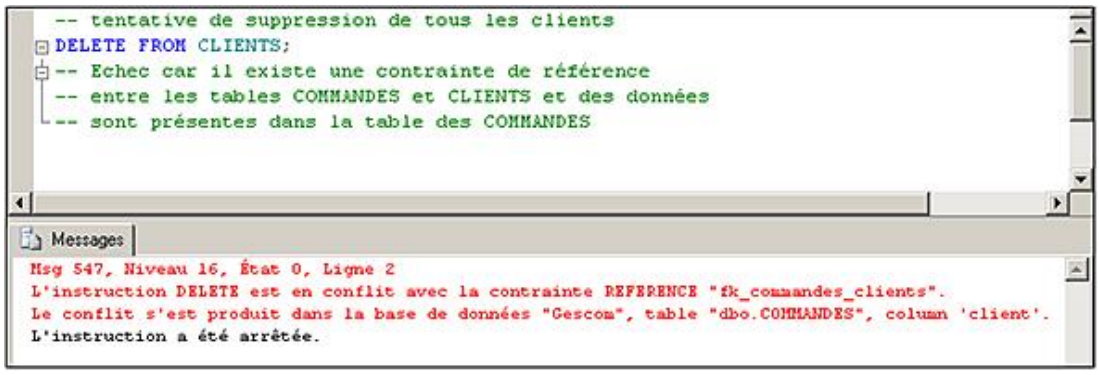
Permet d'utiliser les colonnes d'autres tables ou vues dans la clause WHERE.

`cond`

Expression booléenne permettant de restreindre les lignes à supprimer.

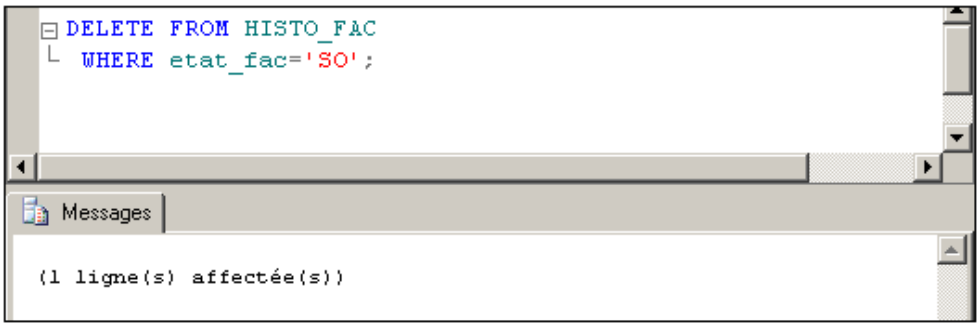
*Exemples*

*Tentative de suppression de TOUS les clients :*

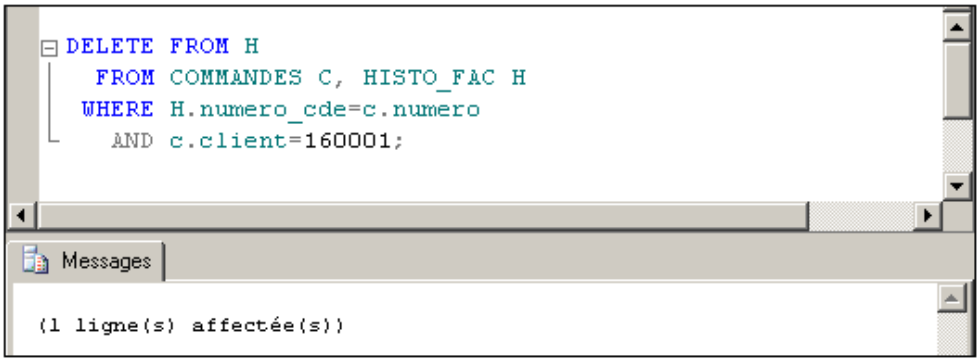



L'option ON DELETE lors de la création de la contrainte de référence permet de contourner ce problème.

*Suppression des factures soldées :*




*Suppression de l'historique du client 160001 :*




## 4. Extraction de lignes

L'instruction SELECT permet de visualiser les données stockées dans les bases, d'effectuer des calculs ou des transformations sur ces données, ou de valoriser des tables à partir d'autres tables.

Cette instruction a une syntaxe complexe que l'on peut étudier en trois parties :

- Syntaxe de base permettant la visualisation et les opérations de l'algèbre relationnelle telles que les restrictions, les projections, les produits cartésiens, les jointures, les calculs élémentaires, les calculs d'agrégats et les unions.
- Syntaxe INTO permettant la création d'une nouvelle table à partir du résultat du SELECT.
- Clause COMPUTE permettant la génération de lignes contenant des statistiques. Cette clause n'est pas relationnelle.

**Syntaxe de base**

```
SELECT [ALL|DISTINCT]{*|liste_expressions}FROM liste_objet[WHERE
cond][GROUP BY liste_expressions][HAVING cond][ORDER BY liste_
expression][UNION[ALL]SELECT.....]
```

`ALL`

Permet l'extraction de toutes les lignes (option par défaut).

`DISTINCT`

N'affiche pas les doublons, c'est-à-dire les lignes résultantes identiques.

`*`

Extrait toutes les colonnes.

`liste_expressions`

Liste composée de noms de colonnes, constantes, fonctions, expressions calculées ou de toute combinaison d'expressions séparées par des virgules. Chaque expression pourra être complétée par un titre de colonne sous la forme : TITRE = exp ou exp ALIAS_COL, afin de modifier le titre par défaut du résultat qui est, soit nul, soit le nom de la colonne dans la table.

`liste_objet`

Liste de tables ou de vues séparées par des virgules, à partir desquelles les données seront extraites. Chaque nom d'objet pourra être suivi d'un nom d'alias permettant de citer la table ou la vue sous un autre nom. De plus, on peut orienter le fonctionnement interne de la requête en plaçant des directives de requêtes pour l'optimiser, entre parenthèses, derrière chaque nom d'objet.

## 5. Opérations de l'algèbre relationnelle

### a. Sélection de colonnes

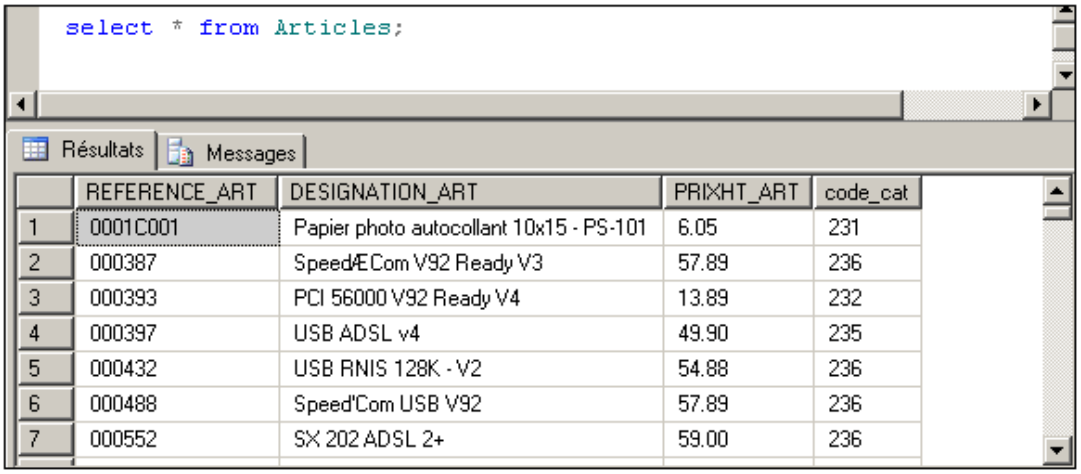
Le mot clé SELECT permet d'introduire une liste de colonnes à extraire des tables dans l'ordre choisi ou toutes les colonnes dans l'ordre de création avec * (ou nom_table.*).

En cas d'ambiguïté dans les noms de colonnes, on doit utiliser la forme : nom_table.nom_col.

La colonne résultante aura le même nom que la colonne initiale sauf si on emploie un titre ou un alias de colonne.

*Exemples*

*Affichage de toutes les colonnes de toutes les lignes de la table Articles :*




| | REFERENCE_ART | DESIGNATION_ART | PRIXHT_ART | code_cat |
|---|---|---|---|---|
| 1 | 0001C001 | Papier photo autocollant 10x15 - PS-101 | 6.05 | 231 |
| 2 | 000387 | Speed/ECom V92 Ready V3 | 57.89 | 236 |
| 3 | 000393 | PCI 56000 V92 Ready V4 | 13.89 | 232 |
| 4 | 000397 | USB ADSL v4 | 49.90 | 235 |
| 5 | 000432 | USB RNIS 128K - V2 | 54.88 | 236 |
| 6 | 000488 | Speed'Com USB V92 | 57.89 | 236 |
| 7 | 000552 | SX 202 ADSL 2+ | 59.00 | 236 |

*Sélection de 2 colonnes :*

```
select code_cat,REFERENCE_ART from Articles
```

Résultats | Messages

| | code_cat | REFERENCE_ART |
|---|---|---|
| 1 | 231 | 0001C001 |
| 2 | 236 | 000387 |
| 3 | 232 | 000393 |
| 4 | 235 | 000397 |
| 5 | 236 | 000432 |
| 6 | 236 | 000488 |
| 7 | 236 | 000552 |

*Changement du titre :*

```
select Catégorie=code_cat,
       'Désignation de l''article' = designation_art
  from Articles;
```

Résultats | Messages

| | Catégorie | Désignation de l'article |
|---|---|---|
| 1 | 231 | Papier photo autocollant 10x15 - PS-101 |
| 2 | 236 | SpeedÆCom V92 Ready V3 |
| 3 | 232 | PCI 56000 V92 Ready V4 |
| 4 | 235 | USB ADSL v4 |
| 5 | 236 | USB RNIS 128K - V2 |
| 6 | 236 | Speed'Com USB V92 |
| 7 | 236 | SX 202 ADSL 2+ |

*Alias de colonne :*

```
select code_cat Catégorie,
       designation_art 'Désignation de l''article'
  from Articles;
```

Résultats | Messages

| | Catégorie | Désignation de l'article |
|---|---|---|
| 1 | 231 | Papier photo autocollant 10x15 - PS-101 |
| 2 | 236 | SpeedÆCom V92 Ready V3 |
| 3 | 232 | PCI 56000 V92 Ready V4 |
| 4 | 235 | USB ADSL v4 |
| 5 | 236 | USB RNIS 128K - V2 |
| 6 | 236 | Speed'Com USB V92 |
| 7 | 236 | SX 202 ADSL 2+ |

Il est également possible de renommer une colonne en utilisant le mot clé **AS** entre le nom de la colonne et son alias dans la clause SELECT. Le résultat observé sera le même que celui présenté dans l'exemple ci-dessus.

### b. Restriction

La restriction consiste à n'extraire qu'un certain nombre de lignes répondant à une ou plusieurs conditions. La clause WHERE permet la mise en œuvre des restrictions. Les conditions sont des expressions booléennes composées de nom de colonnes, de constantes, de fonctions, d'opérateurs de comparaison et des opérateurs logiques.

*Exemples*

*Clients de NANTES :*

```
SELECT numero, nom, prenom, ville
  FROM CLIENTS
 WHERE ville='Nantes';
```

Résultats | Messages

| | numero | nom | prenom | ville |
|---|---|---|---|---|
| 1 | 160428 | anonyme | Édith | Nantes |
| 2 | 160438 | Lampron | Crescent | Nantes |
| 3 | 160512 | Cuillerier | Roland | Nantes |
| 4 | 160996 | Marcil | Aubrey | Nantes |
| 5 | 161325 | Collin | Virginie | Nantes |
| 6 | 161428 | Dodier | Pamella | Nantes |
| 7 | 161454 | Monrency | Germain | Nantes |

*Clients de Loire-Atlantique :*

```
SELECT numero, nom, prenom, ville
  FROM CLIENTS
 WHERE codePostal between 44000 and 44999;
```

Résultats | Messages

| | numero | nom | prenom | ville |
|---|---|---|---|---|
| 1 | 160111 | Fréchette | Jewel | Orvault |
| 2 | 160126 | Perillard | Marie | Rezé |
| 3 | 160265 | Auberjonois | Martine | Rezé |
| 4 | 160324 | Chalut | Norris | Vertou |
| 5 | 160376 | Compagnon | Aurore | Vertou |
| 6 | 160438 | Lampron | Crescent | Nantes |
| 7 | 160454 | David | Clarice | Orvault |

*Clients dont le prénom commence par "ER" et dont la quatrième lettre du nom est "C" et qui habitent dans une ville inconnue ou dont le nom commence par "N" :*

```
SELECT numero, nom, prenom, ville
  FROM CLIENTS
 WHERE (ville like 'N%' OR ville is null)
   AND prenom like 'ER%' and nom like '___C%';
```

Résultats | Messages

| | numero | nom | prenom | ville |
|---|---|---|---|---|
| 1 | 177786 | Marcil | Ernest | Nice |
| 2 | 192827 | Foucault | Ernest | Nancy |

*Commandes passées en Janvier dans les 3 dernières années :*

```
SELECT numero, client, date_cde
  FROM COMMANDES
 WHERE datepart(mm, date_cde)=1
   AND datediff (year, getdate(), date_cde)<3;
```

Résultats | Messages

| | numero | client | date_cde |
|---|---|---|---|
| 1 | 1371 | 161371 | 2008-01-31 23:40:33.0800000 |
| 2 | 1451 | 161451 | 2008-01-30 23:40:33.1100000 |
| 3 | 1463 | 161463 | 2008-01-31 23:40:33.1270000 |

## c. Calculs élémentaires

L'utilisation d'expressions calculées grâce à des opérateurs arithmétiques ou des fonctions, permet d'obtenir des nouvelles colonnes avec le résultat de ces calculs effectués pour chaque ligne résultante.

*Exemples*

*Simulation d'une augmentation de 10 % des tarifs :*

```
SELECT REFERENCE_ART,
       'Ancien prix'=PRIXHT_ART,
       'Nouveau prix'=PRIXHT_ART*1.1
  FROM ARTICLES;
```

Résultats | Messages

| | REFERENCE_ART | Ancien prix | Nouveau prix |
|---|---|---|---|
| 1 | 0001C001 | 6.05 | 6.655 |
| 2 | 000387 | 57.89 | 63.679 |
| 3 | 000393 | 13.89 | 15.279 |
| 4 | 000397 | 49.90 | 54.890 |
| 5 | 000432 | 54.88 | 60.368 |
| 6 | 000488 | 57.89 | 63.679 |

*Affichage des clients avec nom et prénom concaténés et le numéro du département :*

```
SELECT rtrim(nom)+' '+rtrim(prenom) Patronyme,
       substring(convert(char(5), codePostal),1,2) Département
  FROM CLIENTS;
```

Résultats | Messages

| | Patronyme | Département |
|---|---|---|
| 1 | DUPONT Jean | 44 |
| 2 | Brunault Jérôme | 42 |
| 3 | Savard Brie | 14 |
| 4 | Parrot Beaufort | 80 |
| 5 | Rouze Pensee | 69 |
| 6 | Poissonnier Mavise | 75 |

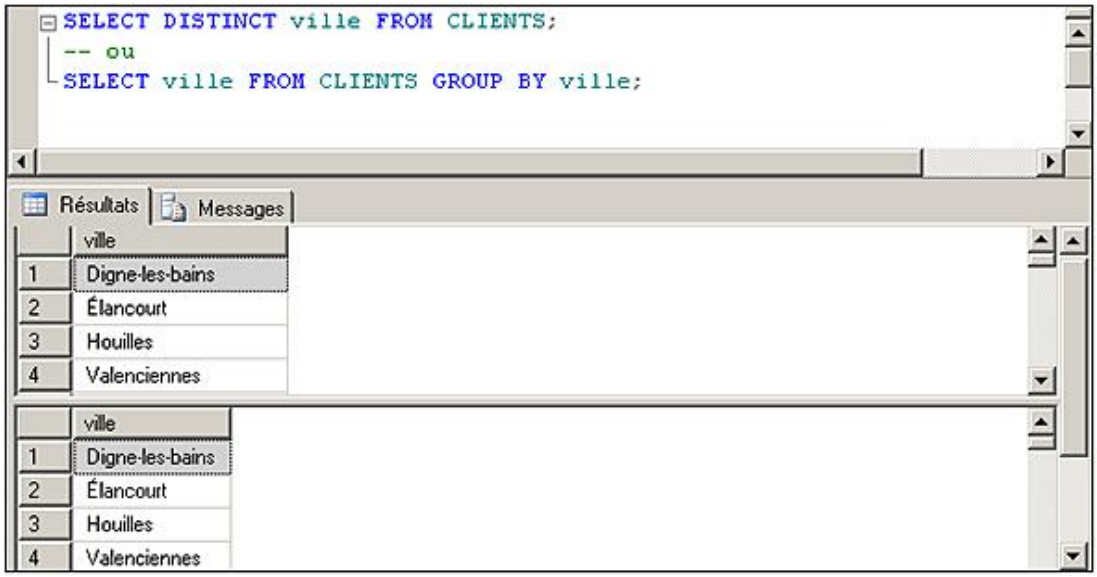
## d. Projection

L'opération de projection permet de regrouper des lignes par rapport à des colonnes. Le résultat de cette opération permettra d'obtenir des lignes uniques sur une sélection de colonnes. La projection s'effectue à l'aide de l'option DISTINCT ou de la clause GROUP BY liste_col. La liste de colonnes ou d'expressions de regroupement doit être rigoureusement identique à la liste de colonnes sélectionnées.

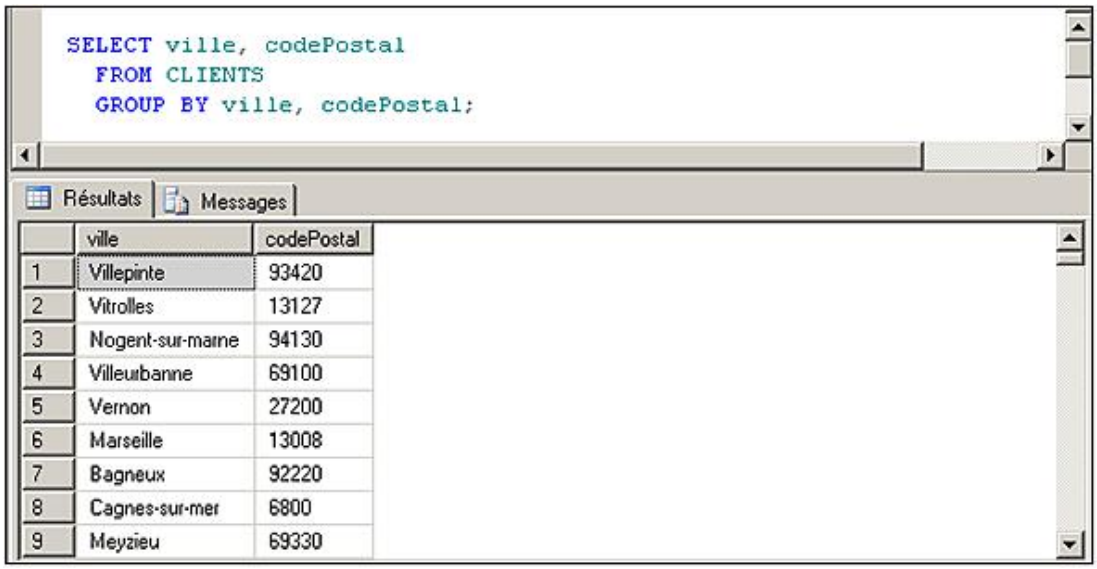
Même si ces deux instructions permettent d'obtenir un résultat similaire, le traitement effectué est lui complètement différent. Avec l'option DISTINCT les informations sont affichées dans l'ordre d'extraction de la base et seules des informations distinctes les unes des autres sont affichées. Dans le cas de la clause GROUP BY, toutes les informations sont extraites de la base puis regroupées (triées) par rapport au critère de regroupement spécifié. Des sous-ensembles sont ainsi constitués et seules les étiquettes de ces sous-ensembles sont affichées.

*Exemples*

*Liste des Villes où habitent les clients :*




*Regroupement sur ville et code postal (pour voir les villes qui ont plusieurs codes postaux) :*




## e. Calculs d'agrégats

Les calculs statistiques portant sur des regroupements de lignes ou sur toute une sélection sont faits en utilisant les fonctions statistiques (COUNT, SUM, AVG, MIN, MAX) et éventuellement la clause GROUP BY. La clause HAVING permet de tester une condition pour limiter les groupes résultats.

*Exemples*

*Prix Minimum, moyen et maximum des articles :*

```
SELECT Prix_mini=MIN(PRIXHT_ART),
       Prix_moyen=AVG(PRIXHT_ART),
       Prix_maxi=MAX(PRIXHT_ART)
  FROM ARTICLES;
```

Résultats | Messages

| | Prix_mini | Prix_moyen | Prix_maxi |
|---|---|---|---|
| 1 | 1.21 | 219.277361 | 7449.00 |

*Quantité d'articles en stock tous dépôts confondus :*

```
SELECT article, sum(quantite) Quantité
  FROM STOCKS
GROUP BY article;
```

Résultats | Messages

| | article | Quantité |
|---|---|---|
| 1 | 0001C001 | 842 |
| 2 | 000387 | 772 |
| 3 | 000393 | 259 |
| 4 | 000397 | 387 |
| 5 | 000432 | 988 |

*Nombre de clients par département, si le département comporte plus de 10 clients :*

```
SELECT Département=substring(convert(char(5), codepostal),1,2),
       nombre=count(*)
  FROM CLIENTS
GROUP BY substring(convert(char(5), codepostal),1,2)
HAVING count(*)>10;
```

Résultats | Messages

| | Département | nombre |
|---|---|---|
| 1 | 39 | 145 |
| 2 | 76 | 829 |
| 3 | 69 | 1412 |
| 4 | 78 | 1337 |
| 5 | 73 | 145 |

## f. Produit cartésien

**Syntaxes**

*Syntaxe courante*

```
SELECT liste_colonne FROM liste_table
```

*Syntaxe ANSI*

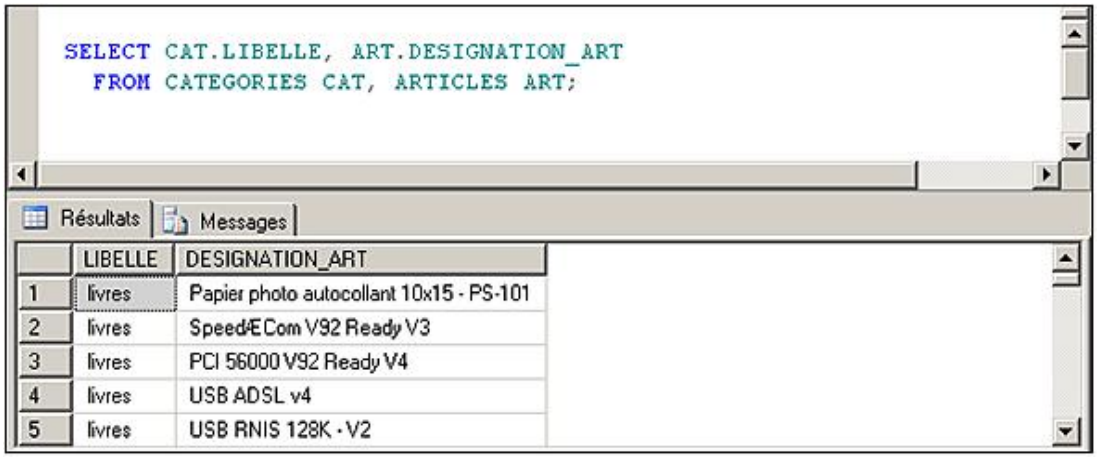
```
SELECT liste_colonne FROM nomtable CROSS JOIN nomtable [ .... ]
```

Le produit cartésien permet d'extraire des données de plusieurs tables en associant chaque ligne de chaque table citée. Les tables concernées doivent être séparées par des virgules derrière le FROM. Si on veut citer le même nom de colonne venant de deux tables différentes, celui-ci doit être précédé du nom de table ou du nom d'alias.

Cette opération peut être utilisée pour des simulations d'association de données ou pour générer un grand nombre de lignes (le nombre de lignes résultantes sera le produit du nombre de lignes de chaque table).

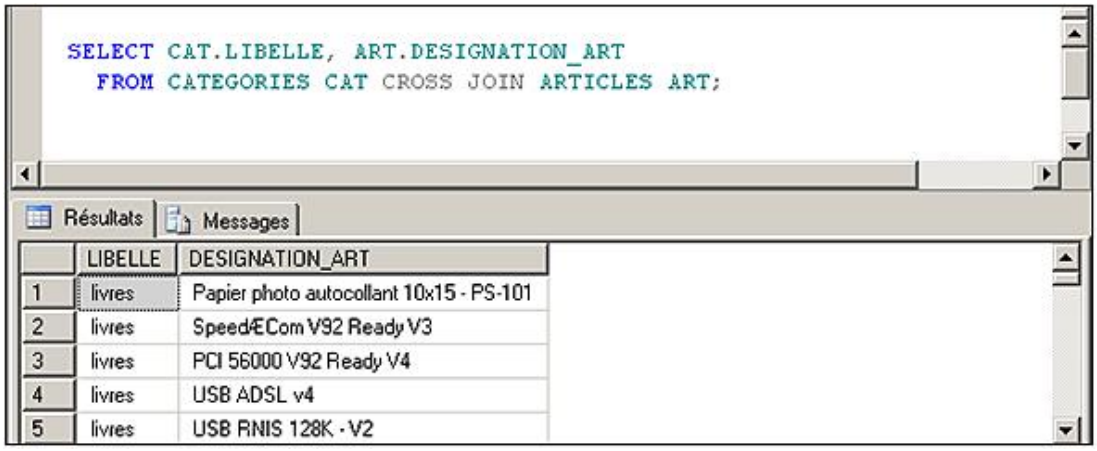
*Exemples*

*Association de chaque article à chaque catégorie :*




| | LIBELLE | DESIGNATION_ART |
|---|---|---|
| 1 | livres | Papier photo autocollant 10x15 - PS-101 |
| 2 | livres | SpeedÆCom V92 Ready V3 |
| 3 | livres | PCI 56000 V92 Ready V4 |
| 4 | livres | USB ADSL v4 |
| 5 | livres | USB RNIS 128K - V2 |

*Même requête en syntaxe ANSI :*




| | LIBELLE | DESIGNATION_ART |
|---|---|---|
| 1 | livres | Papier photo autocollant 10x15 - PS-101 |
| 2 | livres | SpeedÆCom V92 Ready V3 |
| 3 | livres | PCI 56000 V92 Ready V4 |
| 4 | livres | USB ADSL v4 |
| 5 | livres | USB RNIS 128K - V2 |

## g. Jointure

La jointure est la combinaison d'un produit cartésien et d'une restriction. Elle permet d'associer logiquement des lignes de tables différentes. Les jointures sont généralement utilisées pour mettre en correspondance les données d'une ligne comportant une clé étrangère avec les données de la ligne comportant la clé primaire (jointure naturelle).

**Syntaxes**

*Syntaxe courante*

```
SELECT liste_colonne FROM liste_table WHERE
nomtable.nomcolonne opérateur nomtable.nomcolonne [...]
```

*Syntaxe ANSI*

```
SELECT liste_colonne FROM nomtable INNER JOIN nomtable ON
nomtable.nomcolonne opérateur nomtable.nomcolonne [...]
```

*Exemples*

*Visualisation des données articles et catégories :*

```
SELECT CAT.CODE, CAT.LIBELLE, ART.DESIGNATION_ART
  FROM CATEGORIES CAT, ARTICLES ART
 WHERE ART.CODE_CAT=CAT.code;
```

Résultats | Messages

| | CODE | LIBELLE | DESIGNATION_ART |
|---|---|---|---|
| 1 | 231 | Accessoires Barebones | Papier photo autocollant 10x15 - PS-101 |
| 2 | 236 | Adaptateurs | SpeedÆCom V92 Ready V3 |
| 3 | 232 | Accessoires boîtier | PCI 56000 V92 Ready V4 |
| 4 | 235 | Accessoires écrans | USB ADSL v4 |
| 5 | 236 | Adaptateurs | USB RNIS 128K - V2 |

*Même requête en syntaxe ANSI :*

```
SELECT CAT.CODE, CAT.LIBELLE, ART.DESIGNATION_ART
  FROM CATEGORIES CAT INNER JOIN ARTICLES ART
    ON ART.CODE_CAT=CAT.CODE;
```

Résultats | Messages

| | CODE | LIBELLE | DESIGNATION_ART |
|---|---|---|---|
| 1 | 231 | Accessoires Barebones | Papier photo autocollant 10x15 - PS-101 |
| 2 | 236 | Adaptateurs | SpeedÆCom V92 Ready V3 |
| 3 | 232 | Accessoires boîtier | PCI 56000 V92 Ready V4 |
| 4 | 235 | Accessoires écrans | USB ADSL v4 |
| 5 | 236 | Adaptateurs | USB RNIS 128K - V2 |

*Visualisation de la commande :*

```
SELECT CDE.NUMERO,
       convert (char(10), CDE.DATE_CDE,103) as date,
       substring(NOM,1,10) NOM,
       ART.REFERENCE_ART, PRIXHT_ART, QUANTITE
  FROM CLIENTS CLI INNER JOIN COMMANDES CDE
    ON CDE.CLIENT=CLI.NUMERO
 INNER JOIN LIGNES_CDE LIG
    ON LIG.COMMANDE=CDE.NUMERO
 INNER JOIN ARTICLES ART
    ON ART.REFERENCE_ART=LIG.ARTICLE;
```

Résultats | Messages

| | NUMERO | date | NOM | REFERENCE_ART | PRIXHT_ART | QUANTITE |
|---|---|---|---|---|---|---|
| 1 | 1387 | 08/04/2008 | Desilets | 0033S0312 | 227.90 | 8 |
| 2 | 1393 | 30/04/2008 | Sciverit | 0761345-08146-7 | 124.49 | 9 |
| 3 | 1405 | 15/03/2008 | Poulin | 0958A002 | 22.90 | 5 |
| 4 | 1407 | 17/04/2008 | Loiselle | 104530 | 19.99 | 2 |
| 5 | 1362 | 08/05/2008 | Achin | 1394-PCI-3PLUS1 | 11.95 | 9 |
| 6 | 1374 | 09/04/2008 | Couet | 1590B001 | 228.90 | 7 |
| 7 | 1399 | 06/03/2008 | Bonsaint | 1710490-002 | 184.49 | 6 |

## h. Jointure externe

Lorsque la condition n'est pas satisfaite, aucune des lignes n'apparaît dans le résultat. Les jointures externes permettent d'extraire des lignes d'une des deux tables concernées même si la condition est fausse. Dans ce cas, les données de la deuxième table ont la valeur NULL.

La syntaxe utilisée pour la condition est :

```
nom_table1 LEFT OUTER JOIN nom_table2 ON nom_table1.col1=
nom_table2.col2
```

ou

```
nom_table1 RIGHT OUTER JOIN nom_table2 ON nom_table1.col1=
nom_table2.col2
```

ou

```
nom_table1 FULL OUTER JOIN nom_table2 ON nom_table1.col1=nom_table2.col2
```

selon que l'on souhaite voir les lignes de la première (LEFT) ou de la deuxième table (RIGHT).

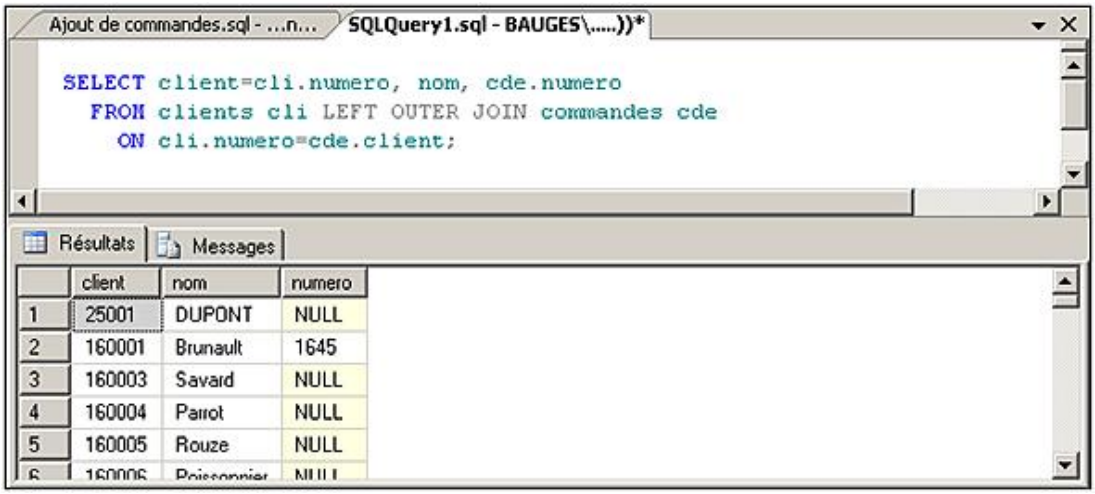
La jointure externe complète (FULL OUTER JOIN) permet quant à elle d'afficher les données issues des deux tables même s'il n'est pas possible d'établir de correspondance.

**Syntaxe**

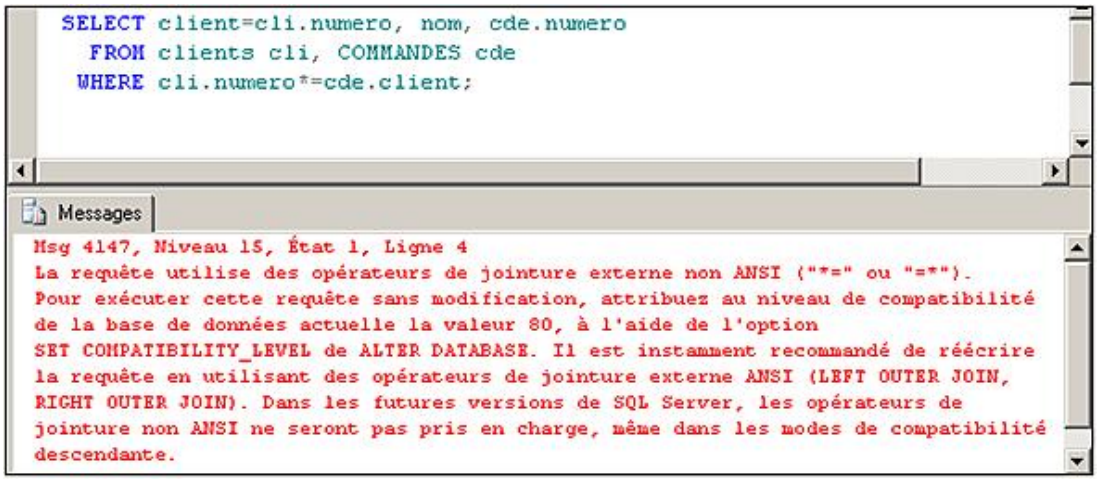
```
SELECT liste_colonne FROM nomtable {LEFT|RIGHT|FULL}OUTER JOIN nomtable
ON nomtable.nomcolonne opérateur nomtable.nomcolonne [...]
```

*Exemple*

*Liste des commandes par client. Les clients n'ayant pas de commande apparaissent quand même :*




La syntaxe SQL Server *= et =* pour définir les jointures externes n'est plus supportée dans SQL Server 2008, comme le montre l'exemple ci-dessous :




La lecture du message d'erreur permet de comprendre l'origine de l'erreur et donne une possibilité pour solutionner le problème.

#### Auto-jointure

Il est possible d'associer des lignes d'une table à d'autres lignes de la même table en réalisant une auto-jointure. L'utilisation des alias de table est alors obligatoire pour éviter les ambiguïtés de syntaxe.

Une auto-jointure a lieu entre deux tables lorsque la condition de jointure correspond à l'égalité et porte sur deux colonnes qui possèdent le même nom.

### i. Order By

La clause Order By permet de trier les données issues d'une requête de type SELECT. Elle permet de préciser selon quelles colonnes les données seront triées. Pour chaque critère de tri, il faut préciser si on utilise l'ordre croissant (par défaut) ou l'ordre décroissant.

Les colonnes sont définies, soit par leur nom, soit par leur numéro d'ordre dans la clause SELECT.

En manipulant la colonne par son numéro d'ordre, il est possible d'effectuer des tris sur des colonnes présentant le résultat d'un calcul élémentaire ou bien d'agrégat.

*Exemple*

*Les clients sont affichés par ordre alphabétique de leur nom et prénom et par codes postaux décroissants.*

```
SELECT nom, prenom, adresse, codePostal, ville
  FROM CLIENTS
ORDER BY nom, prenom, codepostal, ville;
```

Résultats | Messages

| | nom | prenom | adresse | codePostal | ville |
|---|---|---|---|---|---|
| 1 | Abril | Agate | 89 Chemin Challet | 59800 | Lille |
| 2 | Abril | Alain | 38 Chemin Du Lavarin Sud | 6800 | Cagnes-sur-mer |
| 3 | Abril | Aleron | 82 rue Descartes | 67000 | Strasbourg |
| 4 | Abril | Alice | 55 rue Reine Elisabeth | 77000 | Melun |
| 5 | Abril | Anaïs | 38 rue Banaudon | 69005 | Lyon |
| 6 | Abril | Antoine | 41 Rue Marie De Médicis | 11000 | Carcassonne |
| 7 | Abril | Archard | 23 Rue Hubert de Lisle | 56100 | Lorient |

## j. Union

L'opérateur UNION permet d'obtenir un ensemble de lignes provenant de plusieurs requêtes. Toutes les requêtes doivent fournir le même nombre de colonnes, de même type.

*Exemple*

*Affichage des lignes de commande et des quantités en stocks :*

```
SELECT origine='Stocks',convert(char(6), depot) "Commande/dépot", article, quantite
  FROM STOCKS
UNION
SELECT 'Commande',convert(char(6), numero), article, quantite
  FROM LIGNES_CDE lig INNER JOIN COMMANDES cde
    ON lig.commande=cde.numero
ORDER BY article, origine;
```

Résultats | Messages

| | origine | Commande/dépot | article | quantite |
|---|---|---|---|---|
| 1 | Stocks | P2 | 0001C001 | 842 |
| 2 | Stocks | P2 | 000387 | 772 |
| 3 | Stocks | P2 | 000393 | 259 |
| 4 | Stocks | P2 | 000397 | 387 |
| 5 | Stocks | P2 | 000432 | 988 |
| 6 | Stocks | P2 | 000488 | 370 |
| 7 | Stocks | P2 | 000552 | 308 |
| 8 | Stocks | P2 | 0012016SE | 221 |
| 9 | Stocks | P2 | 001241970 | 590 |

## k. Except

Cet opérateur permet de mettre en pratique dans SQL Server, l'opérateur de différence défini en algèbre relationnelle. Il permet de localiser les lignes d'informations présentes dans un jeu de résultats et qui ne le sont pas dans un autre.

Cette différence ne peut être réalisée qu'entre des jeux de résultats possédant la même structure, c'est-à-dire le même nombre de colonnes, définies dans le même ordre et sur les mêmes types de données pour les deux jeux de résultats.

*Exemple*

*Afficher la liste des clients qui habitent dans une ville dont le nom commence par Nan et qui ne sont pas localisés en Loire-Atlantique (44).*

```
SELECT * FROM CLIENTS
WHERE VILLE LIKE 'nan%'
EXCEPT
SELECT * FROM CLIENTS
WHERE CODEPOSTAL BETWEEN 44000 AND 44999;
```

Résultats | Messages

| | numero | nom | prenom | adresse | codepostal | ville | telephone | CODEREP |
|---|---|---|---|---|---|---|---|---|
| 1 | 160172 | Clément | Thibaut | 42 place Stanislas | 54100 | Nancy | 03.35.21.74.31 | NULL |
| 2 | 160228 | Trépanier | Leverett | 1 place Stanislas | 54100 | Nancy | 03.04.51.91.90 | NULL |
| 3 | 160290 | Guérette | Christophe | 55 place Stanislas | 54100 | Nancy | 03.99.71.50.49 | NULL |
| 4 | 160403 | Fluet | Orva | 25 place Stanislas | 54000 | Nancy | 03.07.26.24.31 | NULL |
| 5 | 160428 | anonyme | Édith | NULL | 91220 | Nantes | 01.61.66.92.38 | NULL |
| 6 | 160475 | Tanguay | Searlait | 14 place Stanislas | 54000 | Nancy | 03.56.55.58.79 | NULL |
| 7 | 160746 | Riquier | Oriel | 63 place Stanislas | 54100 | Nancy | 03.11.55.85.93 | NULL |
| 8 | 160772 | Laliberté | Somerville | 45 place Stanislas | 54000 | Nancy | 03.74.11.43.86 | NULL |
| 9 | 160840 | Jolicoeur | Methena | 74 place Stanislas | 54000 | Nancy | 03.32.83.43.02 | NULL |

## l. Intersect

Cet opérateur correspond à la traduction en Transact SQL de l'opérateur ensembliste d'intersection. Il va ainsi être possible d'identifier en une seule requête SQL les lignes d'informations qui sont présentes de façon simultanées dans deux jeux de résultats distincts mais de même structure.

Comme pour l'union et la différence, l'intersection ne peut être mise en place qu'entre des jeux de résultats de même structure.

*Exemple*

*Afficher la liste des clients qui habitent dans une ville dont le nom commence par Nantes et qui sont localisés en Loire-Atlantique (44) :v*

```
SELECT * FROM CLIENTS
WHERE VILLE  like 'nantes%'
INTERSECT
SELECT * FROM CLIENTS
WHERE CODEPOSTAL BETWEEN 44000 AND 44999;
```

Résultats | Messages

| | numero | nom | prenom | adresse | codepostal | ville | telephone | CODEREP |
|---|---|---|---|---|---|---|---|---|
| 1 | 160438 | Lampron | Crescent | 5 place Stanislas | 44100 | Nantes | 02.28.35.72.53 | NULL |
| 2 | 160512 | Cuillerier | Roland | 89 place Stanislas | 44000 | Nantes | 02.83.30.34.66 | NULL |
| 3 | 160996 | Marcil | Aubrey | 14 rue de Raymond Poincaré | 44200 | Nantes | 02.85.31.81.50 | NULL |
| 4 | 161325 | Collin | Virginie | 39 place Stanislas | 44000 | Nantes | 02.44.19.78.55 | NULL |
| 5 | 161428 | Dodier | Pamella | 84 rue de Raymond Poincaré | 44200 | Nantes | 02.86.50.27.58 | NULL |
| 6 | 161454 | Monvency | Germain | 70 place Stanislas | 44000 | Nantes | 02.73.18.00.98 | NULL |
| 7 | 161601 | Fluet | Vick | 52 rue de Raymond Poincaré | 44200 | Nantes | 02.22.97.14.56 | NULL |
| 8 | 161701 | Beaulieu | Vachel | 25 place Stanislas | 44100 | Nantes | 02.02.40.93.24 | NULL |
| 9 | 161762 | Boisclair | Grégoire | 5 place Stanislas | 44100 | Nantes | 02.08.86.78.89 | NULL |

Il existe d'autres moyens pour aboutir au même résultat mais l'objectif est d'illustrer simplement l'opérateur INTERSECT.

## m. Extraire seulement les premières lignes

La clause TOP permet d'extraire seulement les premières lignes d'un jeu de résultats. Cette clause est disponible pour les instructions SELECT, INSERT, UPDATE et DELETE.

**Syntaxe**

```
SELECT TOP (nombre)[PERCENT][WITH TIES] listeColonne
```

`FROM listeTables...`

`nombre`

Représente le nombre de lignes retournées. Il s'agit alors d'un nombre entier (bigint), ou du pourcentage de lignes à ramener. Ce pourcentage peut être exprimé sous forme de float.

`PERCENT`

Spécifie que le nombre représente un pourcentage.

`WITH TIES`

La sélection des lignes à afficher s'effectue après le tri des données par la clause ORDER BY. Il n'est possible d'utiliser cette option que si une clause ORDER BY est précisée dans la requête.

---

Dans la clause SELECT, les parenthèses encadrant le nombre de lignes ou le pourcentage exprimé par TOP sont optionnelles, mais il est très fortement recommandé de les utiliser.

---

*Exemple*

*Les commandes sont triées par chiffres d'affaires décroissants et on ne souhaite connaître que les trois premières. Dans ce premier exemple, la clause WITH TIES n'est pas utilisée et donc la sélection des trois lignes à afficher est effectuée avant l'opération de tri :*

```
SELECT TOP(3) cde.numero, ca=SUM(quantite*prixht_art)
  FROM commandes cde INNER JOIN lignes_cde lig
    ON cde.numero=lig.commande
 INNER JOIN articles art
    ON art.reference_art=lig.article
GROUP BY cde.numero
ORDER BY 2 DESC;
```

Résultats | Messages

| | numero | ca |
|---|---|---|
| 1 | 1633 | 45131.52 |
| 2 | 1445 | 35994.29 |
| 3 | 1392 | 28090.64 |

*Dans ce second exemple, la clause PERCENT permet de connaître 5 % du résultat final :*

```
SELECT TOP 5 PERCENT cde.numero, ca=SUM(quantite*prixht_art)
  FROM commandes cde INNER JOIN lignes_cde lig
    ON cde.numero=lig.commande
 INNER JOIN articles art
    ON art.reference_art=lig.article
GROUP BY cde.numero
ORDER BY 2 DESC;
```

Résultats | Messages

| | numero | ca |
|---|---|---|
| 1 | 1633 | 45131.52 |
| 2 | 1445 | 35994.29 |
| 3 | 1392 | 28090.64 |

---

Dans ce cas, le jeu de résultat contient 20 lignes (information présente en bas à droite de l'écran).

## 6. Requête de création de tables

Il est possible de créer une nouvelle table à partir d'une requête en utilisant la syntaxe :

```
SELECT ..... INTO nom_table FROM .....
```

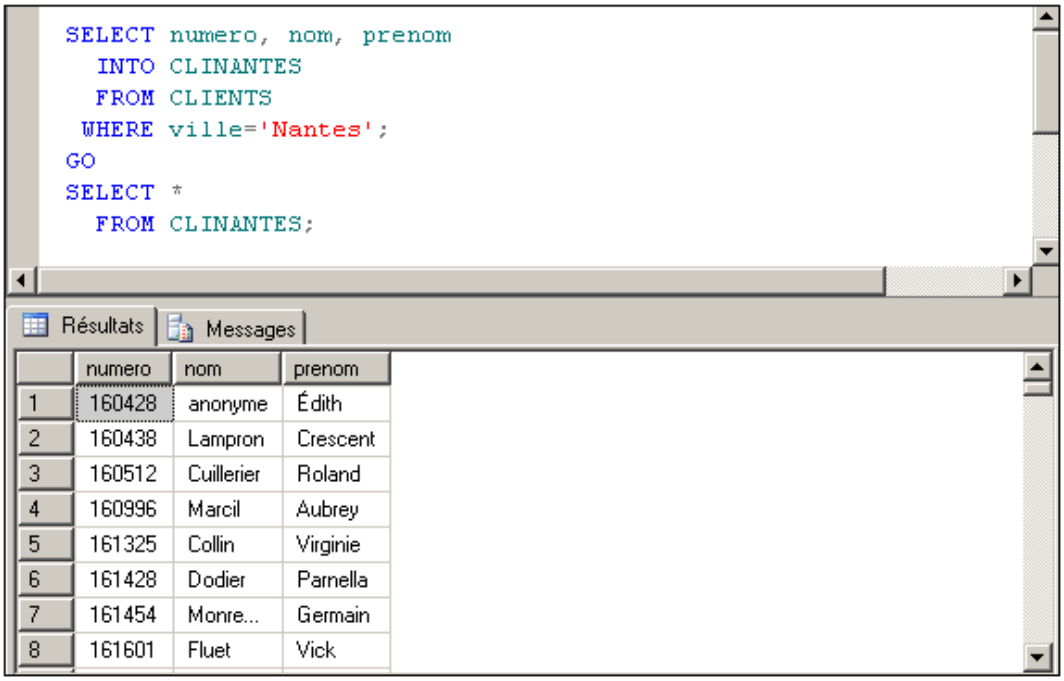
La nouvelle table aura le schéma correspondant aux colonnes extraites. Dans le cas de colonnes calculées, un nom d'alias ou un titre devra être précisé.

Si le nom de la table est précédé de #, la table sera temporaire locale. S'il est précédé de ##, ce sera une table temporaire globale, ces deux types de tables étant stockées dans la base tempdb.

Une table temporaire locale n'est accessible que par la session qui l'a créée et disparaît à la déconnexion. Une table temporaire globale est accessible par toutes les sessions et est supprimée à la fin de la dernière session qui l'a utilisée.

*Exemples*

*Création d'une nouvelle table dans la base courante :*




*Création d'une table temporaire globale :*

```
SELECT article, SUM(quantite) as quantite
  INTO ##totstk
  FROM STOCKS
GROUP BY article;
GO
SELECT *
  FROM ##totstk;
```

Résultats | Messages

| | article | quantite |
|---|---|---|
| 1 | 0001C001 | 842 |
| 2 | 000387 | 772 |
| 3 | 000393 | 259 |
| 4 | 000397 | 387 |
| 5 | 000432 | 988 |
| 6 | 000488 | 370 |
| 7 | 000552 | 308 |
| 8 | 0012016SE | 221 |

## 7. Forcer l'optimiseur de requête

Le langage SQL est un langage interprété qui permet de décrire le résultat que l'on souhaite obtenir. Il est donc relativement facile d'obtenir une description valide du résultat. Bien entendu, pour un même résultat il peut exister différentes façons de le décrire.

À partir de cette description (requête SELECT), l'optimiseur décide du meilleur chemin à utiliser pour fournir le résultat.

La clause OPTION de la requête SELECT permet de spécifier à l'optimiseur de requête la façon dont il doit dresser le plan d'exécution de la requête. Toutefois, comme le volume de données bouge sans cesse et que des améliorations de structure peuvent être apportées (en définissant des index par exemple), cette solution est à utiliser avec parcimonie. Dans tous les cas, il est préférable de laisser à l'optimiseur de requête le soin de dresser le plan d'exécution. En effet, la quasi-totalité du temps, l'optimiseur trouve le meilleur plan d'exécution possible. Les directives à destination de l'optimiseur de requête doivent être utilisées en dernier recours et mises en place par un développeur expérimenté ou bien par l'administrateur de la base de données qui possède une vue globale de l'organisation des données dans la base.

## 8. Tables CTE

Les tables CTE (*Common Table Expression*) ont pour objectif de simplifier l'écriture et donc la compréhension des requêtes. Une table CTE peut être considérée comme une table temporaire et spécifique à une instruction du SQL DML. Une alternative à l'utilisation de tables CTE peut être de définir une table temporaire locale (#matable) avant la requête du SQL DML et à supprimer cette table temporaire locale immédiatement après l'exécution de la requête. Bien évidemment cette alternative est beaucoup plus lourde à gérer et donc moins propre en terme de programmation.

Les tables CTE permettent d'écrire de façon simple des requêtes complexes en simplifiant considérablement l'écriture de requêtes imbriquées.

Les tables CTE peuvent être utilisées dans le cadre d'une requête d'extraction de données (SELECT) mais également pour les requêtes de modification de données (INSERT, UPDATE ou bien DELETE).

---

Les CTE sont des éléments de la norme ANSI SQL 99 ou SQL 3.

---

**Syntaxe**

```
WITH nomTableCTE(nomColonne1, nomColonne2, ...) AS
(
requêteSelect
)
```

La table CTE est créée en utilisant l'instruction WITH suivie du nom de la table accompagné de la liste des colonnes. Enfin, la requête SELECT qui sert de base à la construction de la table CTE est définie après le mot clé AS.

*Exemple de mise en place d'une table CTE*

```
WITH CLI44 AS(
   SELECT numero, nom, prenom
     FROM Clients
     WHERE codepostal BETWEEN 44000 AND 44999)
SELECT * FROM CLI44;
```

Résultats | Messages

| | numero | nom | prenom |
|---|---|---|---|
| 1 | 160111 | Fréchette | Jewel |
| 2 | 160126 | Perillard | Marie |
| 3 | 160265 | Auberjonois | Martine |
| 4 | 160324 | Chalut | Norris |
| 5 | 160376 | Compagnon | Aurore |
| 6 | 160438 | Lampron | Crescent |
| 7 | 160454 | David | Clarice |
| 8 | 160512 | Cuillerier | Roland |

Une fois définie, la table CTE doit être utilisée immédiatement, sinon une erreur est levée.

```
WITH CLI44 AS(
   SELECT numero, nom, prenom
     FROM Clients
     WHERE codepostal BETWEEN 44000 AND 44999)

SELECT COUNT(*) FROM Clients;

SELECT * FROM CLI44;
```

Résultats | Messages

```
(1 ligne(s) affectée(s))
Msg 208, Niveau 16, État 1, Ligne 8
Nom d'objet 'CLI44' non valide.
```

Les tables CTE sont donc plus faciles à créer que les tables temporaires, car l'effort de syntaxe est moindre, par contre le nombre de restrictions est plus large :

- Elles doivent être utilisées immédiatement après leur définition.
- Il n'est pas possible d'utiliser les instructions COMPUTE, ORDER BY, INTO, FOR XML, FOR BROWSE.

## 9. Génération de lignes statistiques

La clause COMPUTE crée des nouvelles lignes contenant des résultats statistiques à partir de fonctions d'agrégation. Cette clause permet de visualiser à la fois les lignes détail et les calculs d'agrégats, à l'inverse de GROUP BY qui ne visualise que les calculs.

**Syntaxe**

```
SELECT.....COMPUTE fonction_stat_ligne (col) [,...]
```

```
[BY col[,...]
```

```
fonction_stat_ligne(col)
```

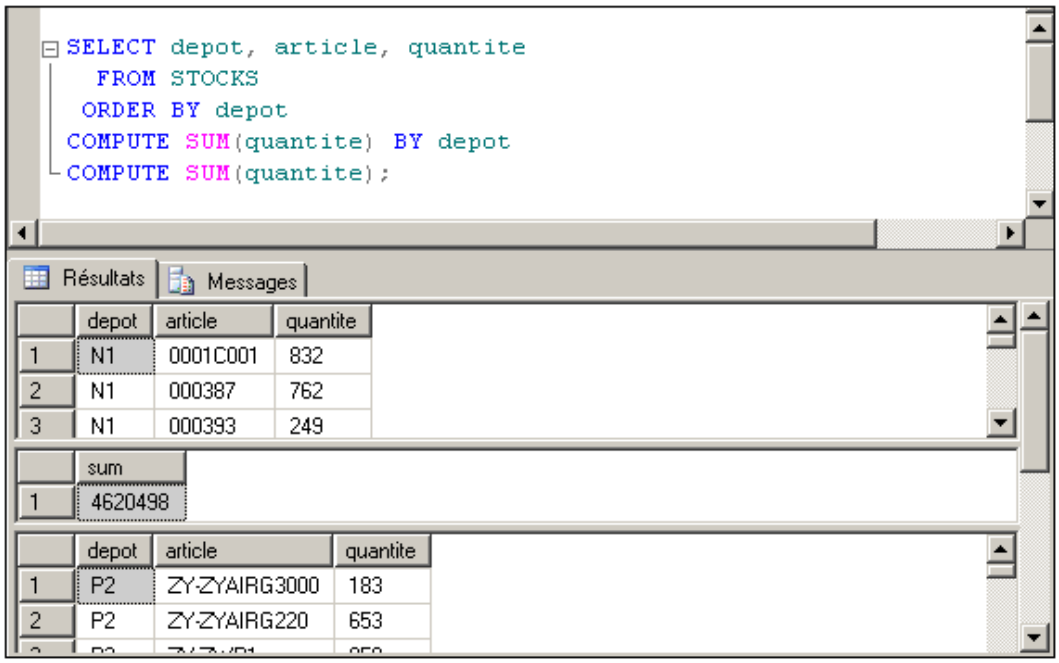
Fonctions COUNT, SUM, AVG, MAX, MIN.

```
BY col
```

Affiche le résultat au changement de valeur de colonne. Dans ce cas, une clause ORDER BY doit être utilisée sur la ou les colonnes concernées.

*Exemple*

*Quantités en stock par Dépôt :*




## Opérateurs ROLLUP et CUBE

Les opérateurs ROLLUP et CUBE sont utilisés en combinaison avec la clause GROUP BY et les fonctions statistiques, afin d'obtenir des lignes supplémentaires comportant le calcul de la fonction, pour des regroupements combinés.

La clause WITH ROLLUP permet de créer des lignes comportant les résultats statistiques pour les regroupements des colonnes du GROUP BY combinées de la gauche vers la droite. Par exemple, si on demande la somme pour un regroupement sur les colonnes A, B et C, la clause WITH ROLLUP fournira, en plus, la somme pour un regroupement sur A, la somme pour un regroupement sur A et B, et la somme totale.

La clause WITH CUBE permet de créer des lignes supplémentaires pour toutes les combinaisons de regroupement des colonnes du GROUP BY. Pour l'exemple précédent, on aura, en plus, la somme pour un regroupement sur B, la somme pour un regroupement sur C, la somme pour un regroupement sur A et C ainsi que la somme pour un regroupement sur B et C.

---

On peut utiliser au maximum 10 expressions de regroupement pour une taille totale de 900 octets.

---

*Exemple*

*Calcul des Quantités en stock totales par dépôt, catégorie et article pour 2 catégories et 2 dépôts :*

```
SELECT depot, libelle, article, SUM(quantite) total
  FROM STOCKS stk
 INNER JOIN ARTICLES art
    ON stk.article=art.reference_art
 INNER JOIN CATEGORIES cat
    ON art.code_cat=cat.code
 WHERE depot IN ('N1','P2')
GROUP BY depot, libelle, article;
```

Résultats | Messages

| | depot | libelle | article | total |
|---|---|---|---|---|
| 1 | N1 | Accessoires Barebones | 0001C001 | 832 |
| 2 | P2 | Accessoires Barebones | 0001C001 | 842 |
| 3 | N1 | Adaptateurs | 000387 | 762 |
| 4 | P2 | Adaptateurs | 000387 | 772 |

*Si on ajoute à la syntaxe la clause WITH ROLLUP, le résultat devient :*

```
SELECT depot, libelle, article, SUM(quantite) total
  FROM STOCKS stk
 INNER JOIN ARTICLES art
    ON stk.article=art.reference_art
 INNER JOIN CATEGORIES cat
    ON art.code_cat=cat.code
 WHERE depot IN ('N1','P2')
GROUP BY depot, libelle, article
WITH ROLLUP;
```

Résultats | Messages

| | depot | libelle | article | total |
|---|---|---|---|---|
| 1... | P2 | Zooms | MA970FA | 905 |
| 1... | P2 | Zooms | NULL | 905 |
| 1... | P2 | NULL | NULL | 46... |
| 1... | NU... | NULL | NULL | 92... |

*Si on remplace WITH ROLLUP par WITH CUBE, le résultat devient :*

```
SELECT depot, libelle, article, SUM(quantite) total
  FROM STOCKS stk
 INNER JOIN ARTICLES art
    ON stk.article=art.reference_art
 INNER JOIN CATEGORIES cat
    ON art.code_cat=cat.code
 WHERE depot IN ('N1','P2')
GROUP BY depot, libelle, article
WITH CUBE;
```

Résultats | Messages

| | depot | libelle | article | total |
|---|---|---|---|---|
| 1 | N1 | Accessoires Barebones | 0001C001 | 832 |
| 2 | P2 | Accessoires Barebones | 0001C001 | 842 |
| 3 | NULL | Accessoires Barebones | 0001C001 | 1674 |
| 4 | NULL | NULL | 0001C001 | 1674 |

## Opérateur OVER

Cette clause permet de partitionner les données ou bien de les trier avant, par exemple, d'appliquer une fonction de calcul d'agrégat ou une fonction de tri, c'est-à-dire ROW_NUMBER, DENSE_RANK, RANK et NTILE.

Dans le cadre d'une fonction de tri, la clause OVER va pouvoir contenir un partitionnement et/ou une instruction ORDER BY pour effectuer un tri. Dans le cas où une fonction de calcul d'agrégat est utilisée, seul le partitionnement des données est possible avec la clause OVER.

### Syntaxe

```
OVER ([PARTITION BY expression, ...][ORDER BY colonne, ...])
```

*Exemple*

```
SELECT depot,art.code_cat,
       libelle, article,designation_art,
       SUM(quantite) OVER (PARTITION BY depot, code_cat) as total
  FROM STOCKS stk
 INNER JOIN ARTICLES art
    ON stk.article=art.reference_art
 INNER JOIN CATEGORIES cat
    ON art.code_cat=cat.code;
```

Résultats | Messages

| | depot | code_cat | libelle | article | designation_art | total |
|---|---|---|---|---|---|---|
| 1 | N1 | 231 | Accessoires Barebones | 0001C001 | Papier photo autocollant 10x15 - PS-101 | 278191 |
| 2 | N1 | 231 | Accessoires Barebones | 0105013 | CÔble KVM classiques - PS2 | 278191 |
| 3 | N1 | 231 | Accessoires Barebones | 0107003 | Rallonge USB 3 m | 278191 |
| 4 | N1 | 231 | Accessoires Barebones | 0108012 | CÔble VGA - 3m | 278191 |

## Opérateur NTILE

Cette fonction est utilisée conjointement à OVER et permet de diviser chaque partition en des groupes de données équilibrés. Par exemple, avec l'instruction NTILE(4) OVER… le résultat de chaque partition sera divisé en 4 groupes. Les données seront réparties de façon équitable dans chaque groupe. Si le nombre de lignes présentes dans la partition n'est pas un multiple du nombre de groupes à créer alors les premier groupes contiennent une ligne de plus que les derniers groupes.

Dans le cas présenté ici, si 4 groupes doivent être créés et que la partition contient 15 lignes alors les 3 premiers groupes vont contenir 4 lignes et le 4$^{ième}$ groupe contiendra 3 lignes.

### Syntaxe

```
NTILE(entier) OVER ...
```

*Exemple*

```
SELECT depot,art.code_cat,
       libelle, article,designation_art,
       NTILE (4) OVER (PARTITION BY depot, code_cat ORDER BY code_cat) as ensemble
  FROM STOCKS stk
 INNER JOIN ARTICLES art
    ON stk.article=art.reference_art
 INNER JOIN CATEGORIES cat
    ON art.code_cat=cat.code;
```

Résultats | Messages

| | depot | code_cat | libelle | article | designation_art | ensemble |
|---|---|---|---|---|---|---|
| 1 | N1 | 231 | Accessoires Barebones | 0001C001 | Papier photo autocollant 10x15 - PS-101 | 1 |
| 2 | N1 | 231 | Accessoires Barebones | 0105013 | CÔble KVM classiques - PS2 | 1 |
| 3 | N1 | 231 | Accessoires Barebones | 0107003 | Rallonge USB 3 m | 1 |
| 4 | N1 | 231 | Accessoires Barebones | 0108012 | CÔble VGA - 3m | 1 |
| 5 | N1 | 231 | Accessoires Barebones | 0109022 | SUB-50 vers DB-25 | 1 |

## 10. Sous-requêtes imbriquées

Il est possible d'imbriquer une requête SELECT dans une autre requête SELECT (ou dans une instruction UPDATE ou DELETE) dans tous les cas d'utilisation d'une expression. En général, l'utilisation se fait avec les clauses WHERE ou HAVING.

On peut distinguer plusieurs types de sous-requêtes :

- sous-requêtes renvoyant une seule valeur. Elles peuvent être utilisées avec les opérateurs =,< ,<= >,> =.
- sous-requêtes renvoyant une liste de valeurs. Elles peuvent être utilisées avec les opérateurs IN, EXISTS, ANY, SOME ou ALL.
- sous-requêtes corrélées (ou subordonnées). La clause WHERE de la requête interne fait référence à une des tables de la requête externe. Dans ce cas, la sous-requête interne est exécutée pour chaque ligne extraite par la requête externe.

*Exemples*

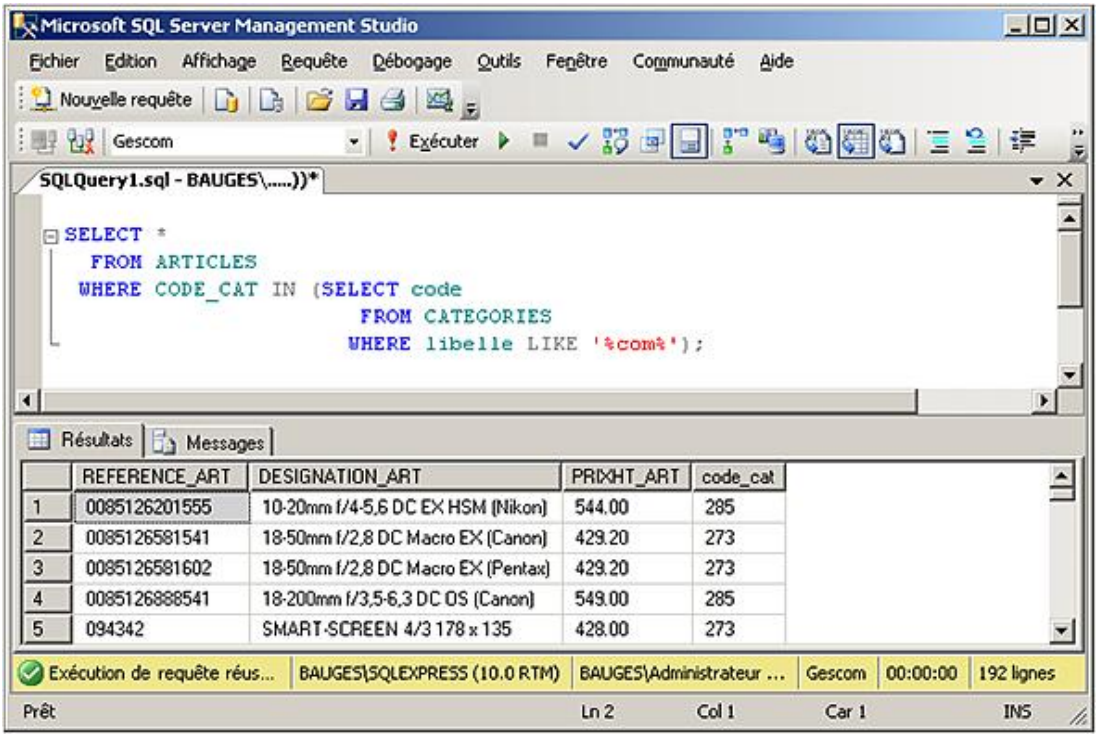
*Commandes du Client DURAND :*

```
SELECT *
  FROM COMMANDES
 WHERE client IN (SELECT numero
                    FROM CLIENTS
                   WHERE nom like 'Durand');
/* Cette requête peut générer une erreur si plusieurs
   cients s'appellent Durand*/
```

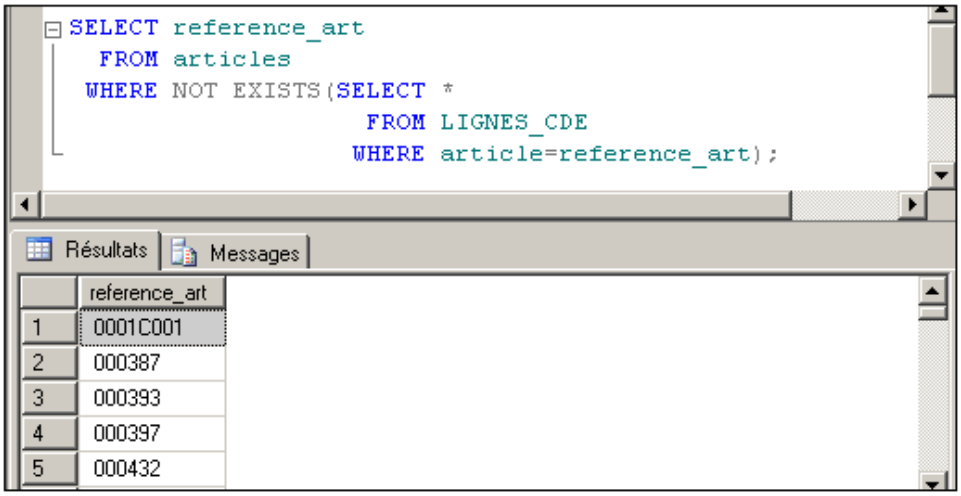
Résultats | Messages

| | numero | date_cde | taux_remise | client | etat |
|---|---|---|---|---|---|
| 1 | 1350 | 2008-02-04 23:40:33.0670000 | NULL | 161350 | NULL |

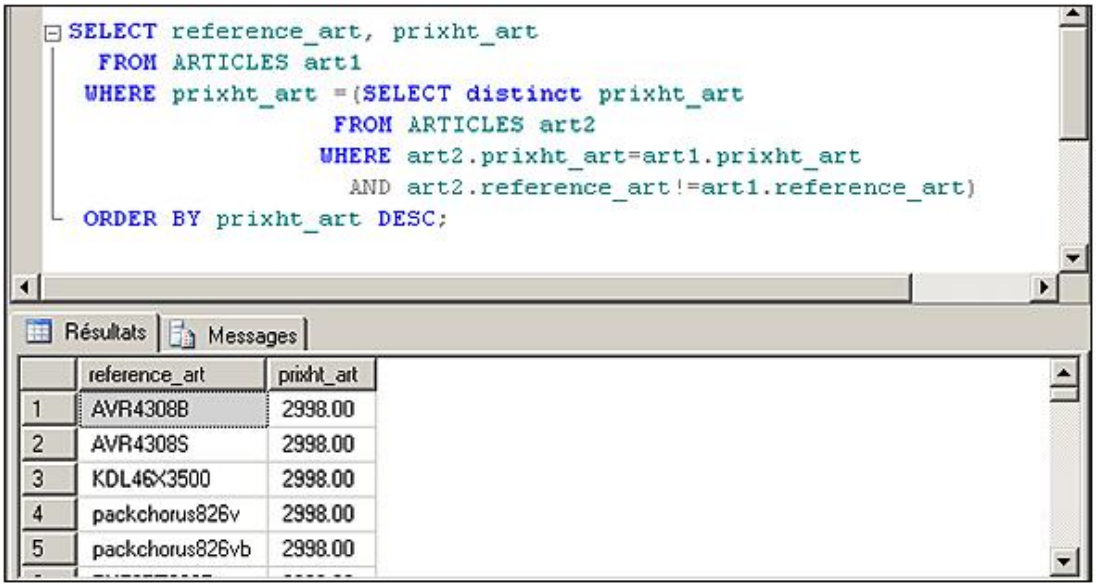
*Extraction des articles dont le libellé de catégorie comporte "COM" :*




*Liste des articles ne faisant partie d'aucune commande :*




*Liste des articles ayant des prix identiques :*




## 11. PIVOT et UNPIVOT

Ces deux instructions du Transact SQL sont puissantes et d'une grande facilité d'emploi. L'objectif de PIVOT est de transformer un résultat présenté sous forme de lignes distinctes en colonnes distinctes. UNPIVOT réalise l'opération inverse.

Le travail réalisé par PIVOT était effectué auparavant en définissant une instruction case compliquée, ou en s'appuyant sur les autres opérateurs tels que la jointure, l'union, le calcul d'agrégat pour présenter l'information comme souhaitée. Dans tous les cas, un codage long est fastidieux et peut être la source de dysfonctionnements.

Avant d'utiliser cette opération, il faut définir quelles sont les données qui vont être concernées et quelle colonne va représenter le PIVOT. En général, l'objectif est de réaliser un tableau de synthèse pour lequel la valeur présente dans la cellule provient d'un calcul d'agrégat dépendant de la ligne et de la colonne.

---

Pour pouvoir exécuter ces instructions le serveur doit être défini en niveau de compatibilité 9.0 (sp_dbcmptlevel) ou 10.0. En effet, ces instructions ont été introduites dans le Transact SQL par SQL Server 2005.

---

### Comment utiliser PIVOT ?

L'instruction PIVOT fait partie de la clause FROM de l'instruction SELECT. L'utilisation de cette instruction va donc permettre de créer une pseudo table qui sera interne à la requête. Comme toutes les tables manipulées dans une requête, il est possible de lui spécifier un alias de table par l'intermédiaire de la clause AS.

Avant de faire appel à la fonction PIVOT, il faut déterminer quel est le calcul d'agrégat à effectuer et pour quelles valeurs de la colonne, il est possible de réaliser ce traitement. La colonne autour de laquelle le pivot est effectué doit posséder une valeur convertible en nvarchar car les valeurs vont se transformer en nom de colonne. Il n'est donc pas possible d'organiser un pivot autour d'une valeur au format binaire.

#### Extrait de syntaxe

```
SELECT
FROM [...]
PIVOT (calculAgregat FOR colonnePivot IN
(listeDeValeur)) as aliasDeTable
```

La liste de valeur permet de préciser les valeurs de la colonne PIVOT qui vont se transformer en colonne. Dans la liste, chaque valeur doit être spécifiée entre crochets [ ] sans apostrophe pour les chaînes de caractères. Cette même notation sera reprise derrière la clause SELECT afin de préciser le titre de la colonne ainsi que l'ordre des colonnes.

L'alias de table pour le PIVOT est indispensable.

---

Lorsque le PIVOT est utilisé avec un calcul d'agrégat alors les valeurs null ne sont pas prises en compte dans la réalisation du calcul. Si l'on souhaite traiter ces valeurs, il faut utiliser la fonction Transact SQL ISNULL.

---

## Exemple d'utilisation de PIVOT

Dans l'exemple suivant, la table des stocks contient une ligne d'information par article et par dépôt. Pour l'instant, il existe trois entrepôts distincts : N1, N2 et P1 qui sont localisés respectivement à Nantes pour les 2 premiers et à Paris pour le troisième.

*Exemple de la table des stocks*

```
use Gescom;
go
SELECT article, depot,quantite FROM STOCKS
```

Résultats | Messages

| | article | depot | quantite |
|---|---|---|---|
| 1 | 0001C001 | N1 | 832 |
| 2 | 0001C001 | N2 | 1674 |
| 3 | 0001C001 | P2 | 842 |
| 4 | 000387 | N1 | 762 |
| 5 | 000387 | N2 | 1534 |
| 6 | 000387 | P2 | 772 |
| 7 | 000393 | N1 | 249 |
| 8 | 000393 | N2 | 508 |

Pour obtenir une vue synthétique de la disponibilité d'un article, il est nécessaire de parcourir trois lignes d'informations. Il n'est donc pas facile de lire l'information.

Par contre, en faisant pivoter les informations par rapport à la référence articles et en affichant en regard de cette dernière, les quantités disponibles en stocks dans chacun des trois dépôts, alors l'information sera plus pertinente.

*Exemple d'extraction en utilisant PIVOT*

```
use Gescom;
go
SELECT article, [N1] as "depot1",[N2] as "depot2", [P2] as "depot3"
  FROM STOCKS
  PIVOT (sum(quantite) FOR depot in([N1],[N2],[P2])) as pvt
```

Résultats | Messages

| | article | depot1 | depot2 | depot3 |
|---|---|---|---|---|
| 1 | 7640A003 | 563 | 1136 | 573 |
| 2 | CGAS002E1B | 763 | 1536 | 773 |
| 3 | MOWI-018 | 168 | 346 | 178 |
| 4 | PR400-004 | 834 | 1678 | 844 |
| 5 | J4899C | 719 | 1448 | 729 |
| 6 | LIVE-V | 863 | 1736 | 873 |
| 7 | THPL | 380 | 770 | 390 |
| 8 | 62000 | 594 | 1198 | 604 |

Afin d'éclaircir le résultat, il est possible de réaliser des jointures avec d'autres tables. Dans l'exemple suivant, une jointure est faite avec la table des articles afin de connaître la désignation des articles.

*Exemple d'utilisation d'une jointure en plus du pivot*

```
use Gescom;
go
SELECT article, [N1] as "depot1",[N2] as "depot2", [P2] as "depot3",DESIGNATION_ART
  FROM STOCKS
  PIVOT (sum(quantite) FOR depot in([N1],[N2],[P2])) as pvt
  INNER JOIN Articles on Articles.REFERENCE_ART=pvt.article;
```

Résultats | Messages

| | article | depot1 | depot2 | depot3 | DESIGNATION_ART |
|---|---|---|---|---|---|
| 1 | 0001C001 | 832 | 1674 | 842 | Papier photo autocollant 10x15 - PS-101 |
| 2 | 000387 | 762 | 1534 | 772 | SpeedÆCom V92 Ready V3 |
| 3 | 000393 | 249 | 508 | 259 | PCI 56000 V92 Ready V4 |
| 4 | 000397 | 377 | 764 | 387 | USB ADSL v4 |
| 5 | 000432 | 978 | 1966 | 988 | USB RNIS 128K - V2 |
| 6 | 000488 | 360 | 730 | 370 | SpeedCom USB V92 |
| 7 | 000552 | 298 | 606 | 308 | SX 202 ADSL 2+ |
| 8 | 001201655 | 211 | 432 | 221 | 1201655 |

### UNPIVOT

La fonction UNPIVOT réalise le travail inverse à celui fait par PIVOT. Cependant son usage est plus restreint car le cas se présente moins fréquemment de vouloir transformer des colonnes en lignes. Si toutefois le cas se présente, alors cette fonctionnalité peut être mise en œuvre à l'aide de la syntaxe suivante :

```
SELECT ... colonneUnpivot, AliasNouvelleColonne
FROM table
UNPIVOT (AliasNouvelleColonne FOR colonneUnPivot IN
listeDeValeur) AS aliasDeTable
```

Comme pour l'utilisation du PIVOT, il est possible de faire des jointures avec une table qui contient la colonne UNPIVOT.

*Exemple d'utilisation d'UNPIVOT*

```
use Gescom;
go
WITH Monstock(article, Nantes1, Nantes2, Paris2, designation_art) as (
SELECT article, [N1] as "depot1",[N2] as "depot2", [P2] as "depot3",DESIGNATION_ART
  FROM STOCKS
  PIVOT (sum(quantite) FOR depot in([N1],[N2],[P2])) as pvt
  INNER JOIN Articles on Articles.REFERENCE_ART=pvt.article)
  SELECT article, designation_art,quantite
    FROM Monstock
    UNPIVOT (quantite FOR Monstock IN (Paris2)) as unpvt;
```

Résultats | Messages

| | article | designation_art | quantite |
|---|---|---|---|
| 1 | 0001C001 | Papier photo autocollant 10x15 - PS-101 | 842 |
| 2 | 000387 | SpeedÆCom V92 Ready V3 | 772 |
| 3 | 000393 | PCI 56000 V92 Ready V4 | 259 |
| 4 | 000397 | USB ADSL v4 | 387 |

Une table CTE est utilisée dans le script afin de faciliter sa compréhension.

## 12. MERGE

L'instruction MERGE permet en une seule opération Transact SQL d'effectuer des modifications, des ajouts et même des suppressions sur une même table de destination. La sélection de l'action à réaliser s'effectue en précisant des critères de sélection. Naturellement, l'instruction MERGE ne peut pas utiliser la même table comme source et destination. Toutefois, il est possible de définir une condition de jointure entre la source et la destination afin de limiter les données manipulées.

Cette instruction permet de simplifier certains traitements de l'information, pour la construction de synthèses par exemple. Sans cette instruction il est nécessaire de définir un lot Transact SQL avec un curseur pour être en mesure de traiter chaque ligne de la source de façon individuelle et de décider ainsi de l'action à effectuer.

**Syntaxe**

```
MERGE [ INTO ] tableCible
USING sourceDonnée
ON <conditionSélection>
[ WHEN MATCHED [ AND <conditionSélection> ]
      THEN <opération> ]
[ WHEN NOT MATCHED [BY TARGET] [ AND < conditionSélection > ]
      THEN <opération> ]
[ WHEN NOT MATCHED [BY SOURCE] [ AND < conditionSélection > ]
      THEN <opération> ]
```

`tableCible`

Il s'agit de la table qui contient les données sur lesquelles vont porter les instructions INSERT, UPDATE et DELETE.

`sourceDonnées`

C'est à partir de cette source de données que sont issues les informations qui permettent de sélectionner le type d'opération à exécuter ainsi que les données à utiliser.

`conditionSélection`

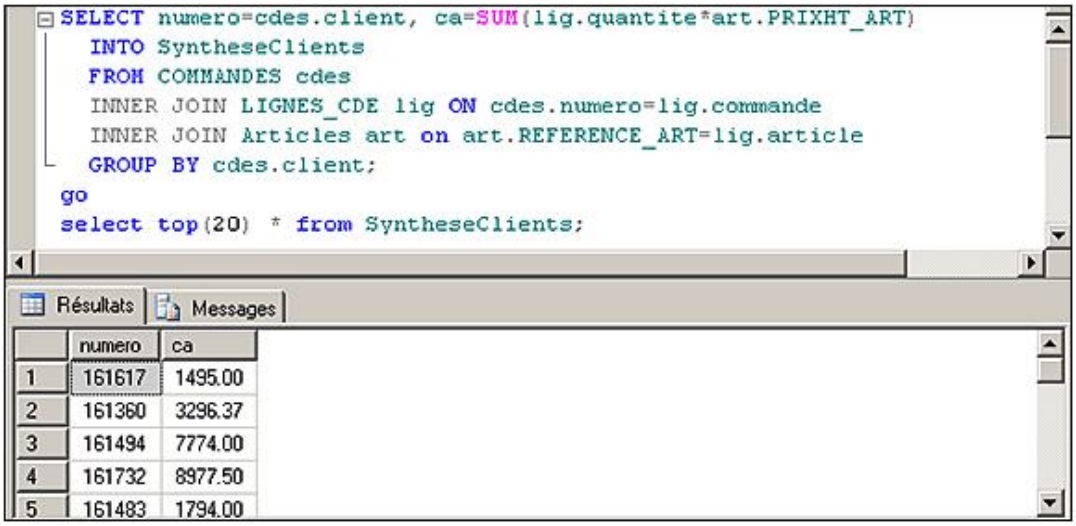
Critère de restriction permettant de savoir si les données doivent être sélectionnées ou bien si l'opération qui suit doit être exécutée.

`opération`

Instruction INSERT, UPDATE ou bien DELETE à exécuter sur la table cible.

*Exemples*

*La table SyntheseClients contient les références de tous les clients qui ont passé au moins une commande, c'est-à-dire qui possèdent un chiffre d'affaires :*




*Une colonne CA de type numeric(10,2) est ajoutée à la table des clients de la façon suivante :*

```
alter table clients
  add ca numeric(10,2) null;
```

```
Messages
Commande(s) réussie(s).
```

*En s'appuyant sur les informations contenues dans la table syntheseClients, la table Clients va être modifiée de la façon suivante :*

- *Mise à jour de la colonne ca avec le chiffre d'affaires calculé.*
- *Mise à zéro du CA pour les autres clients.*

```
merge clients as cli
using syntheseClients as bilan
on cli.numero=bilan.numero
when matched then
  update set ca=bilan.ca
when not matched by source then
  update set ca=0;
```

# Gestion des vues

SQL Server permet la gestion d'objets associés aux tables : les vues (VIEWS). Une vue est une table virtuelle, c'est-à-dire un objet ayant la même utilisation qu'une table par rapport au Langage de Manipulation de Données, à quelques restrictions près, mais n'occupant pas d'espace disque pour les données. Une vue ne "stocke" que la requête correspondant à l'extraction.

Les intérêts d'utilisation d'une vue sont multiples :

- Simplification des structures des tables. Certaines tables peuvent comporter de nombreuses colonnes avec des noms et des types peu pratiques à manipuler. Une vue fournira à l'utilisateur les mêmes données dans une forme simplifiée.

- Réutilisation de requêtes. Lorsque les requêtes sont souvent exécutées (jointures, calculs), une vue permettra de stocker l'instruction et de l'utiliser plus simplement.

- Sécurité d'accès. Il est possible de cacher des lignes et des colonnes aux utilisateurs en ne mettant à leur disposition que des vues de projection ou de restriction à la place des tables initiales.

La modification des données au travers d'une vue n'est autorisée que si une seule table correspondant à la vue est modifiée et si la requête de la vue n'utilise pas de calculs.

## Création de vues

On peut créer une vue par le langage de définition de données ou par SQL Server Management Studio.

### Syntaxe

```
CREATE VIEW nom
[WITH ENCRYPTION | WITH SCHEMABINDING| WITH VIEW_METADATA]
AS requête [WITH CHECK OPTION]
```

`nom`

Nom d'objet, doit être unique dans la base.

`requête`

Instruction SELECT ne comportant pas de clause ORDER BY, UNION, COMPUTE ou INTO.

`WITH ENCRYPTION`

Permet de crypter le code dans les tables système.

`WITH SCHEMABINDING`

Permet de lier la vue au schéma. Avec une telle option, les objets référencés dans la vue doivent être nommés de la façon suivante, *nom_schema.nom_ objet*, et les tables utilisées dans la vue ne peuvent être supprimées. De même, si une opération ALTER TABLE affecte la définition de la vue, alors elle échoue.

`WITH VIEW_METADATA`

Permet de préciser à SQL Server de renvoyer les informations de métadonnées correspondant à la vue et non pas celles des tables qui composent la vue. Cette demande d'informations de métadonnées est particulièrement importante dans le cadre de l'accès aux données via une interface de programmation comme ODBC ou OLE-DB.

`WITH CHECK OPTION`

Permet de ne pas autoriser l'insertion ni la modification de données ne répondant pas aux critères de la requête.

## Suppression de vues

**Syntaxe**

```
DROP VIEW nom
```

*Exemple*

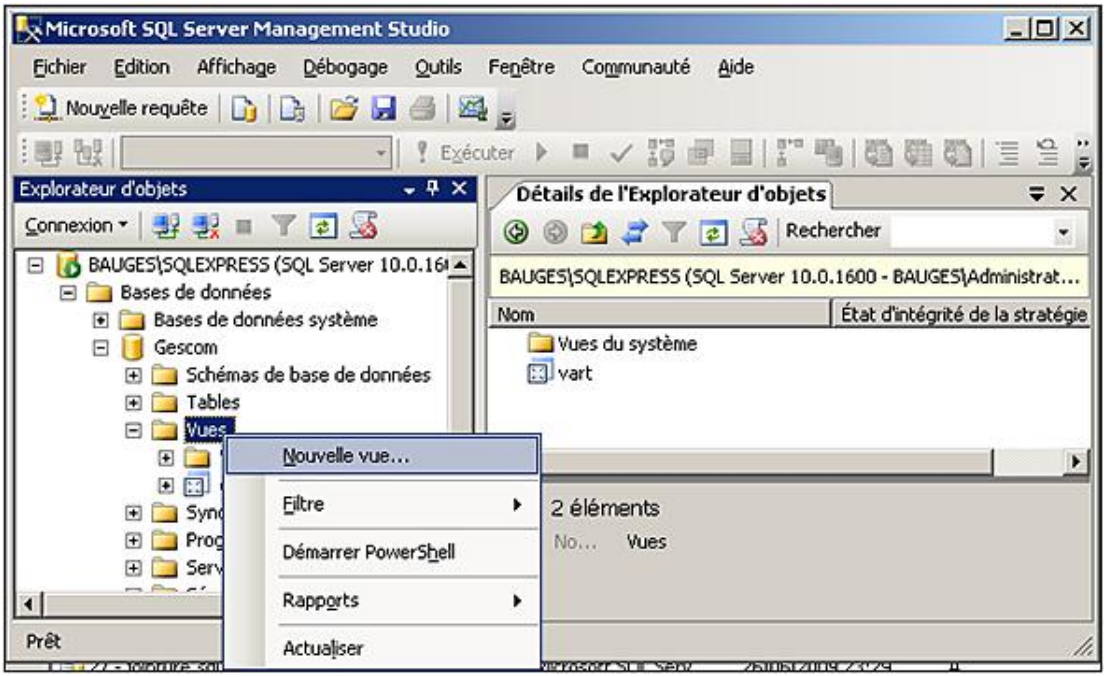
*Simplification de la table articles :*

```
create view vart as
   select ref=CONVERT(char(5), code_cat)+'-'+REFERENCE_ART,
          des=SUBSTRING(DESIGNATION_ART,1,10), prix=PRIXHT_ART
     from Articles;
go
select * from vart;
```

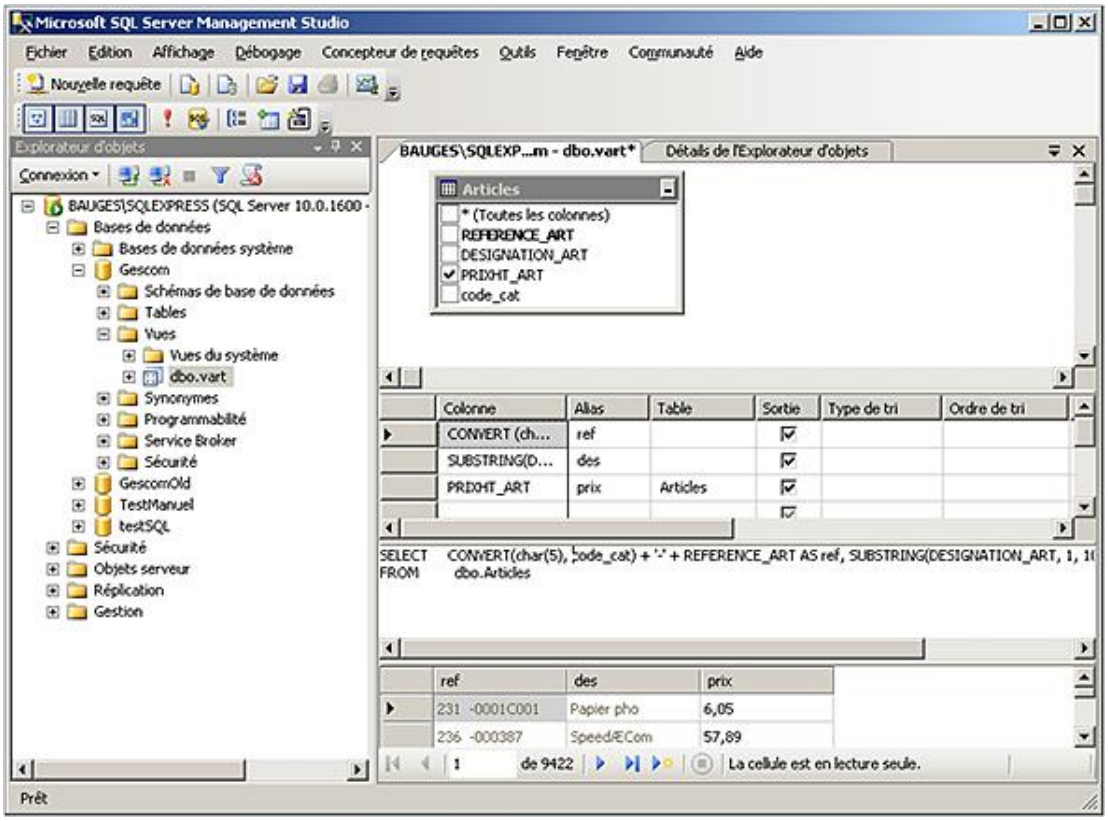
Résultats | Messages

| | ref | des | prix |
|---|---|---|---|
| 1 | 231 -0001C001 | Papier pho | 6.05 |
| 2 | 236 -000387 | SpeedÆCom | 57.89 |
| 3 | 232 -000393 | PCI 56000 | 13.89 |

*Création d'une vue par SQL Server Management Studio :*




*Édition d'une vue par SQL Server Management Studio :*




Les déclencheurs de type INSTEAD OF sont particulièrement bien adaptés pour les vues car ils permettent de détourner les actions INSERT, UPDATE et DELETE vers les tables qui composent la vue. Ces déclencheurs donnent la possibilité d'avoir des vues dont le comportement est totalement translucide pour l'utilisateur de la base de données.

# Le SQL procédural

SQL Server est un serveur de base de données relationnelle et à ce titre, il fournit tous les éléments pour stocker de façon structurée les données mais aussi les outils nécessaires pour travailler avec les données au travers de SQL. Avec le Transact SQL il est également possible de définir des traitements procéduraux directement dans la base de données. Ces traitements vont pouvoir être utilisables par tous les utilisateurs de la base sous réserve qu'ils possèdent les privilèges nécessaires. Il est possible de conserver la définition de ces traitements et de les rendre paramétrables par l'intermédiaire de la création de procédures et de fonctions.

Des traitements procéduraux pourront également être mis en place pour définir des contraintes d'intégrité complexes, il s'agira alors de triggers ou déclencheurs de base de données.

## 1. Gestion des variables utilisateur

Une variable est une zone mémoire, caractérisée par un nom et un type, permettant de stocker une valeur. Les variables Transact SQL doivent obligatoirement être déclarées avant leur utilisation. Elles peuvent ensuite remplacer n'importe quelle expression dans les instructions SQL.

### Déclaration de variables

```
DECLARE @nom_variable type [,...]
```

`nom_variable`

Nom précédé du caractère @.

`type`

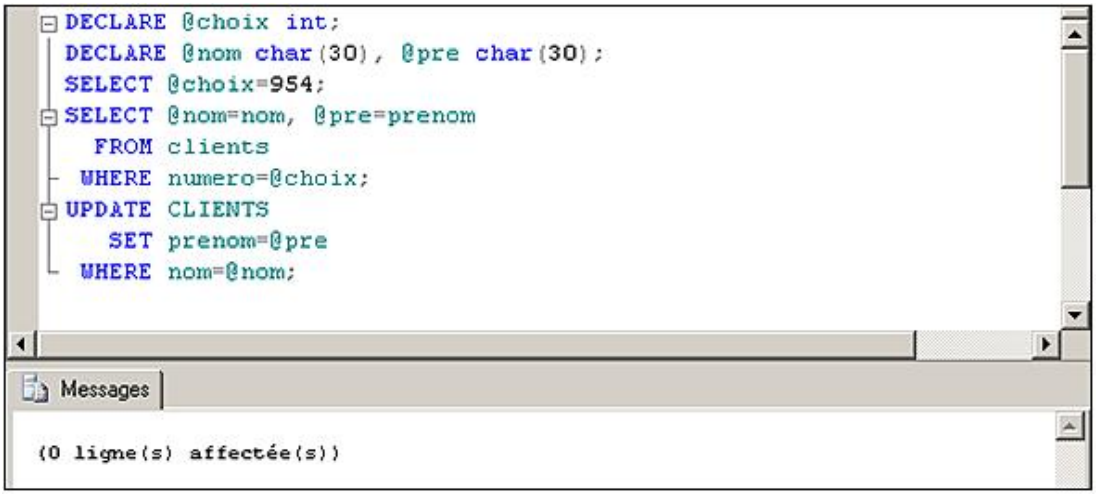
Type système ou défini par l'utilisateur.

### Valorisation des variables

```
SELECT @nom_variable = expr [,...][FROM...]
```

*Exemple*

*Modification du prénom des clients qui ont le même nom que le 954 :*




## 2. Variables système

Ces variables sont définies par le système, et peuvent être utilisées seulement en lecture. Elles se distinguent des variables utilisateur par le double @.

`@@CONNECTIONS`

Nombre de connexions ou de tentatives de connexion depuis le dernier démarrage de SQL Server.

`@@CPU_BUSY`

Temps consacré par l'unité centrale à SQL Server depuis le dernier démarrage de celui-ci. Le résultat est exprimé en unité CPU. Il faut multiplier par @@TIMETICKS pour avoir un résultat en microsecondes.

`@@CURSOR_ROWS`

Nombre de lignes affectées dans le dernier curseur ouvert.

`@@DATEFIRST`

Renvoie la valeur courante du paramètre SET DATEFIRST.

`@@DBTS`

Valeur du type de données timestamp courant pour la base de données.

`@@ERROR`

Dernier numéro d'erreur généré par le système.

`@@FETCH_STATUS`

Contient le statut d'une commande de curseur FETCH.

`@@IDENTITY`

Enregistre la dernière valeur IDENTITY insérée.

`@@IDLE`

Temps, en millisecondes, pendant lequel SQL Server est resté inactif depuis son dernier démarrage.

`@@IO_BUSY`

Temps, en millisecondes, consacré par SQL Server à effectuer des opérations d'entrée/sortie depuis son dernier démarrage.

`@@LANGID`

Identificateur de la langue actuellement utilisée.

`@@LANGUAGE`

Langue actuellement utilisée.

`@@LOCK_TIMEOUT`

Timeout, en millisecondes, de la session en cours.

`@@MAX_CONNECTIONS`

Nombre maximal de connexions simultanées qu'il est possible d'établir avec SQL Server.

`@@MAX_PRECISION`

Renvoie le niveau de précision utilisé par les types de données décimal et numeric.

`@@NESTLEVEL`

Niveau d'imbrication de l'instruction en cours d'exécution.

`@@OPTIONS`

Informations sur les valeurs courantes des options SET.

`@@PACK_RECEIVED`

Nombre de paquets entrants lus par SQL Server depuis son dernier démarrage.

`@@PACK_SENT`

Nombre de paquets sortants écrits par SQL Server depuis son dernier démarrage.

`@@PACKET_ERRORS`

Nombre d'erreurs qui se sont produites alors que SQL Server envoyait ou recevait des paquets depuis son dernier démarrage.

`@@PROCID`

Identificateur de la procédure stockée Transact SQL du traitement en cours d'exécution.

`@@REMSERVER`

Renvoie le nom du serveur contenu dans l'enregistrement des noms d'accès d'un serveur distant.

`@@ROWCOUNT`

Nombre de lignes affectées par la dernière instruction.

`@@SERVERNAME`

Nom du serveur SQL local.

`@@SERVICENAME`

Nom du service en cours d'exécution.

`@@SPID`

Numéro d'identification du processus courant sur le serveur.

`@@TEXTSIZE`

Longueur maximale, en octets, des données texte ou image renvoyées par une instruction SELECT.

`@@TIMETICKS`

Nombre de millisecondes par pulsation.

`@@TOTAL_ERRORS`

Nombre d'erreurs rencontrées par SQL Server en lisant ou en écrivant des données depuis son dernier démarrage.

`@@TOTAL_READ`

Nombre de lectures de données sur le disque effectuées par SQL Server depuis son dernier démarrage.

`@@TOTAL_WRITE`

Nombre d'écritures de données sur le disque effectuées par SQL Server depuis son dernier démarrage.

`@@TRANCOUNT`

Nombre de transactions actuellement actives pour l'utilisateur courant.

`@@VERSION`

Date, numéro de version et type de processeur de la version courante de SQL Server.

## 3. Les transactions

### Gestion des transactions

Le premier point à prendre en considération est le verrouillage des informations. En effet, lorsque SQL Server lit des données ou les modifie, il verrouille les lignes d'informations manipulées. Ce verrouillage dure le temps de l'exécution de l'instruction ou le temps de la transaction.

Suivant le type de verrous posés, il est possible ou non à d'autres transactions d'accéder simultanément à la même information.

Une transaction est un ensemble d'instructions de manipulations de données s'exécutant dans une même unité de travail. La validation d'une transaction assure que toutes les instructions en faisant partie se sont correctement terminées, l'annulation de la transaction assurera l'annulation de l'ensemble des instructions.

Seules les instructions du DML (SELECT, INSERT, UPDATE, DELETE) sont prises en compte dans une transaction. Sont exclues toutes les instructions manipulant des objets (CREATE, ALTER, DROP, SELECT INTO, GRANT, REVOKE, LOAD, DUMP...).

Les transactions sont utiles pour assurer l'intégrité et la cohérence des données lors de modifications multiples, pour améliorer les performances, pour tester les effets de modification, pour gérer les verrouillages.

Une transaction est caractérisée par le mot clé ACID (*Atomicity Consistency Isolation Durability*) qui peut se traduire en français par Atomique Consistance Indépendance Durée.

- Atomique car une transaction représente une unité atomique (non divisible) de travail pour le serveur.
- Consistance car à la fin de la transaction les informations présentes dans la base doivent être consistantes, c'est-à-dire cohérentes par rapport aux règles de structuration des données mises en place.
- Indépendance car les données visibles sont, soit celles d'avant la transaction, soit celles résultantes de la transaction. Il n'est pas possible, depuis une autre transaction, de visualiser les données en cours de modification dans une transaction.
- Durée car lorsqu'une transaction est validée, les changements apportés par la transaction sur les données sont durables et le gestionnaire de base de données doit garantir que, quoi qu'il arrive au niveau du système, ces changements seront toujours visibles.

### Syntaxes

*Début de transaction.*

```
BEGIN TRAN[SACTION][nomtransaction]
```

*Validation de transaction.*

```
COMMIT TRAN[SACTION][nomtransaction]
```

*Déclaration d'un point de contrôle.*

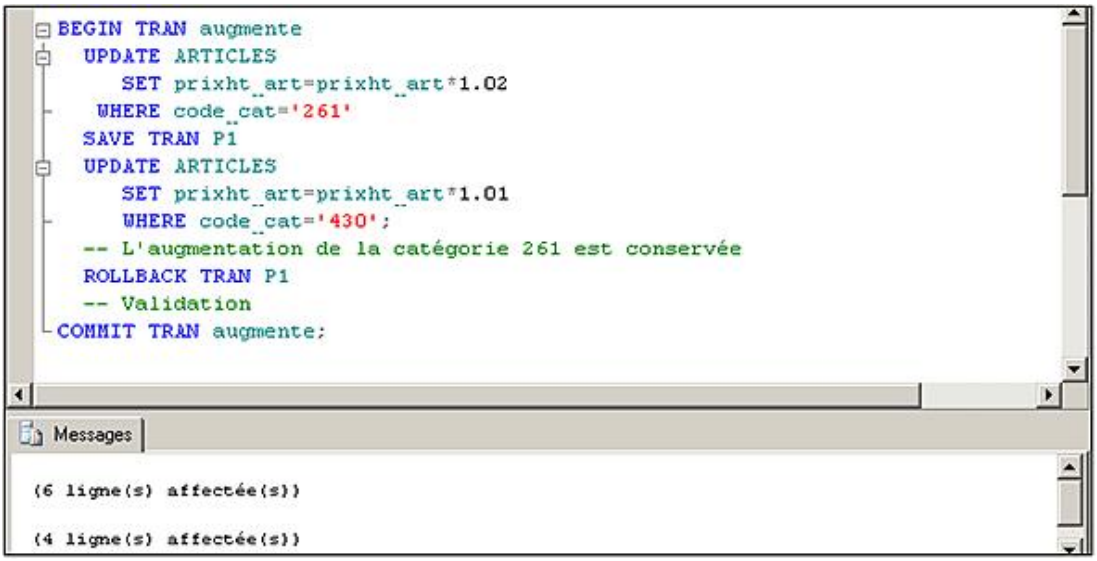
```
SAVE TRAN[SACTION][nom du point de contrôle]
```

*Annulation de transaction.*

```
ROLLBACK TRAN[SACTION][{nomtransaction/nom point de contrôle}]
```

*Exemple*

*Simulation d'augmentation de tarifs :*




## Gestion des verrous

Lors de transactions concurrentes, SQL Server gère automatiquement des verrous afin de garantir la cohérence des données de chaque transaction.

Une transaction ne pourra pas modifier des lignes accessibles par une autre transaction, et ne pourra lire des lignes en cours de modification (lecture cohérente).

SQL Server pose automatiquement les verrous lors de l'exécution des transactions. Ces verrous peuvent être posés à différents niveaux : ligne, pages, tables, extension... Le type de verrou est choisi par SQL Server afin de minimiser les coûts. Les types disponibles sont :

**Verrous partagés**

Destinés aux opérations de lecture, ils empêchent la modification de données. Plusieurs transactions peuvent poser des verrous partagés sur les mêmes éléments.

**Verrous de mise à jour**

Mis en place en vue d'opérations de modifications (UPDATE LOCK). Seule une transaction peut acquérir un verrou de ce type sur une ressource, les autres transactions doivent attendre. Si la transaction ne met pas à jour les données lues, alors le verrou se transforme en verrou partagé.

**Verrous exclusifs**

Destinés aux opérations d'écriture, ils n'autorisent aucun autre verrou. Ils agissent au niveau table ou page.

**Verrous d'intention**

Ils permettent à SQL Server de signaler qu'une transaction souhaite acquérir un verrou de mise à jour ou partagé sur la ressource. La pose de verrous exclusifs est impossible.

Il est possible d'agir sur les verrous de plusieurs façons, au niveau de la configuration, au niveau des transactions ou en obtenant des informations par l'intermédiaire de la vue dynamique **sys.dm_tran_locks** ou dans SQL Server Management Studio et en sélectionnant le nœud **Management - Activity Monitor** dans l'**Object Explorer**.

## SET TRANSACTION

On peut définir le système de verrouillage des instructions SELECT pour toutes les transactions de la session.

**Syntaxe**

```
SET TRANSACTION ISOLATION LEVEL option
```

**Options**

READ UNCOMMITTED

Ce niveau permet de lire des modifications non encore validées effectuées par d'autres transactions. Ce type de lecture est appelé lecture sale ou dirty reads, car les données visualisées ne seront pas forcément conservées en cas d'annulation de transaction (ROLLBACK). Dans ce mode de fonctionnement, les instructions DML ne vont pas réclamer un verrou partagé lors de la lecture des données car rien n'empêche la modification des données lues, même si la modification n'est pas encore validée.

READ COMMITTED

C'est le mode de fonctionnement par défaut. Seules les modifications validées (COMMIT) sont visibles. Les instructions du DML utilisent toujours un verrouillage exclusif. Cependant, les verrous acquis pour la lecture des informations sont relâchés, non pas en fin de transaction mais dès que la lecture est terminée. Dans le cas où la même ligne est lue plus d'une fois au cours de la même transaction, il n'est pas garanti que la lecture de la même ligne retourne toujours le même lot d'informations. Ce cas peut se présenter si les données sont modifiées et validées par une autre transaction entre les deux opérations de lecture.

REPEATABLE READ

Avec ce mode de fonctionnement, les données lues ou modifiées par une transaction ne sont plus accessibles par les autres transactions et ce, afin de garantir que la lecture répétée d'une ligne de données retourne toujours les mêmes informations. Cependant, les informations lues sont encore accessibles en lecture pour les autres transactions. Les verrous sont donc définis pour la durée de la transaction. Avec ce type de verrouillage, des informations peuvent être ajoutées dans les tables entre le début et la fin de la transaction et donc le jeu de données n'est plus le même. Ce type de problème peut avoir une incidence significative lors, par exemple, d'un calcul de remise par rapport à un chiffre d'affaires effectué par une transaction et simultanément une commande supplémentaire est ajoutée par une autre transaction. Cette nouvelle commande vient modifier le résultat du calcul de remise.

SERIALIZABLE

Avec ce mode de fonctionnement, les données lues ou modifiées par une transaction sont accessibles uniquement par cette transaction. Les données simplement lues restent accessibles en lecture seule pour les autres transactions. Les verrous sont définis pour la durée de la transaction au niveau des lignes de données. Des verrous de plus haut niveau, dans les index, sont également définis afin d'éviter l'ajout de données dans le jeu de résultats de la transaction. S'il n'existe pas d'index couvrant la requête de lecture, alors un verrou de plus au niveau est posé, c'est-à-dire un verrouillage au niveau de la table.

## Options de verrouillage

Il est possible de spécifier le verrouillage d'une table pour une instruction SELECT particulière.

Il est très fortement recommandé de laisser la gestion des verrous au moteur relationnel de SQL Server afin d'optimiser cette gestion. La gestion manuelle ne permet que rarement une gestion optimum et peut donc provoquer des problèmes d'interblocage ou de délai d'attente pour les autres transactions.

**Syntaxe**

```
SELECT.....FROM nomtable WITH (option de verrou)
```

**Options de verrou**

NOLOCK

Pas de verrous.

HOLDLOCK

Maintien de verrous partagés jusqu'à la fin de la transaction.

UPDLOCK

Maintien de verrous de mises à jour de pages jusqu'à la fin de la transaction.

TABLOCK

Utilisation de verrous partagés sur la table.

`PAGLOCK`

Utilisation de verrous partagés sur les pages.

`TABLOCKX`

Maintien d'un verrou exclusif.

`ROWLOCK`

Permet de spécifier qu'un verrou de niveau ligne doit être choisi lorsque l'optimiseur de requête aurait plutôt fait le choix d'en acquérir un au niveau de la page ou de la table.

*Exemple*

*Pose d'un verrou partagé pendant l'extraction (niveau table) :*

```
SELECT ville, count(*) 'Nbre Clients'
  FROM CLIENTS WITH (TABLOCK HOLDLOCK)
 GROUP BY ville;
```

Résultats | Messages

| | ville | Nbre Clients |
|---|---|---|
| 1 | Digne-les-bains | 79 |
| 2 | Élancourt | 68 |
| 3 | Houilles | 63 |

On peut activer le verrouillage au niveau ligne lors des insertions (IRL : *Insert Row-level Locking*) en exécutant la procédure stockée :sp_tableoption 'nomtable','insert row lock','true'.

## 4. Gestion des lots et des scripts

Un lot d'instructions est un ensemble d'instructions Transact SQL qui sera compilé et exécuté en une seule unité. Il se termine par la commande GO.

Un lot peut comporter n'importe quelle instruction ou série d'instructions ainsi que des transactions.

L'intérêt des lots réside dans l'amélioration des performances, ainsi que dans la compilation unique. Dans le cas d'une erreur de syntaxe, aucune instruction n'est exécutée.

Cependant, les lots sont soumis à certaines restrictions :

- Il n'est pas possible de combiner certaines instructions dans un même lot : CREATE PROCEDURE, CREATE RULE, CREATE DEFAULT, CREATE TRIGGER, CREATE VIEW.
- Il n'est pas possible d'agir sur des définitions de colonne et d'utiliser ces modifications dans un même lot (valeurs par défaut, contrainte CHECK, ajout de colonnes à une table).
- Il n'est pas possible de supprimer un objet et de le recréer dans un même lot.

Les scripts sont des ensembles de lots qui seront exécutés successivement à partir d'un fichier texte. Ces fichiers ont, par convention, l'extension '.sql'.

## 5. Contrôle de flux

C'est un ensemble de fonctionnalités comportant des instructions (RETURN, RAISERROR, PRINT) et des structures de contrôles (séquence, alternative, répétitive) qui améliorent l'utilisation des instructions Transact SQL en permettant à l'utilisateur de contrôler leur exécution.

### a. RETURN

Cette instruction permet de sortir inconditionnellement d'une procédure ou fonction en renvoyant éventuellement une valeur entière.

**Syntaxe**

```
RETURN [exprn]
```

*Exemple*

*Renvoi d'une valeur d'une procédure pour indiquer le bon fonctionnement :*

```
CREATE PROC p1 AS
  IF (1<2)  --Condition
    RETURN 0;    --Procédure OK
  ELSE
    RETURN 1;    --Erreur
```

Messages

```
Commande(s) réussie(s).
```

*Utilisation :*

```
DECLARE @verif_procedure int;
EXEC @verif_procedure=p1;
```

Messages

```
Commande(s) réussie(s).
```

*On pourra alors tester la variable @verif_procedure.*

### b. PRINT

C'est une instruction d'affichage de message.

**Syntaxe**

```
PRINT {'texte'|@variable|@@variablesystème}
```

*Exemple*

```
DECLARE @nb int;
SELECT @nb=count(*) FROM CLIENTS;
PRINT 'Nombre total de clients';
PRINT @nb;
```

```
Messages
Nombre total de clients
37801
```

### c. CASE

C'est une expression qui permet d'attribuer des valeurs conditionnelles.

**Syntaxe**

```
CASE [expression]
      WHEN {valeur|condition} THEN
      valeurattribuée1
     [...]
     [ELSE valeurattribuée n]
      END
```

Renvoie la valeur attribuée en fonction de la valeur de l'expression ou en fonction d'une condition.

*Exemple*

```
SELECT libelle=CASE code_cat
            WHEN '01' THEN 'Micros'
            WHEN '02' THEN 'Logiciel'
            ELSE 'Divers'
            END,
          reference_art, designation_art
  FROM ARTICLES
 ORDER BY code_cat;
```

Résultats | Messages

| | libelle | reference_art | designation_art |
|---|---|---|---|
| 5 | Divers | SST-FN82 | SST-FN82 |
| 6 | Divers | SJM2600-10 | SJM2600 |
| 7 | Divers | sock7547000A-AIC | Adaptateur socket 7... |
| 8 | Divers | sockA7000A-AICu | Adaptateur socket ... |
| 9 | Divers | SST-PP02 | Power Supply Accu |

L'exemple suivant montre l'utilisation de l'instruction CASE avec une condition de comparaison au niveau de la clause WHEN. Cette condition va fournir un résultat de type booléen.

```
SELECT 'designation prix'=CASE
    WHEN prixht_art<400 THEN 'Promotion'
    WHEN prixht_art BETWEEN 400 AND 800 THEN 'Normal'
    ELSE 'Exclusif'
    END
  FROM ARTICLES
 WHERE code_cat='260';
```

Résultats | Messages

| | designation prix |
|---|---|
| 1 | Promotion |
| 2 | Promotion |
| 3 | Promotion |
| 4 | Promotion |
| 5 | Promotion |

### d. BEGIN... END

C'est une structure de contrôle permettant de délimiter une série d'instructions (bloc). Elle est utilisée avec les tests (IF) et les boucles (WHILE).

**Syntaxe**

```
BEGIN
    {instruction | bloc}
    ...
END
```

### e. IF

C'est la structure de contrôle alternative permettant de tester une condition et d'exécuter une instruction ou un bloc si le test est vrai.

**Syntaxe**

```
IF condition
   {instruction|bloc}
[ELSE]
   {instruction|bloc}
```

*Exemple*

```
DECLARE @nocli int
SELECT @nocli=15689
IF EXISTS(SELECT * FROM CLIENTS WHERE numero=@nocli)
  BEGIN
    DELETE FROM COMMANDES WHERE client=@nocli
    DELETE FROM CLIENTS WHERE numero=@nocli
    PRINT 'Supression OK'
  END
ELSE
  PRINT 'Pas de client pour ce numéro'
```

Messages

```
Pas de client pour ce numéro
```

## f. WHILE

C'est la structure de contrôle répétitive qui permet d'exécuter une série d'instructions tant qu'une condition est vraie.

**Syntaxe**

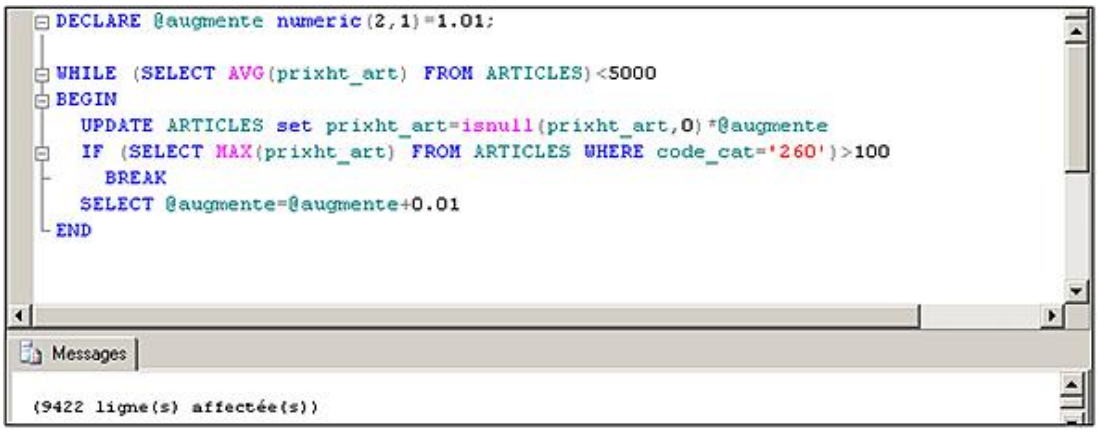
```
WHILE condition
      {instruction|bloc}
```

L'instruction BREAK permet la sortie de la boucle.

L'instruction CONTINUE permet de repartir à la première instruction de la boucle.

*Exemple*

*On augmente les tarifs jusqu'à ce que le prix d'un article de la catégorie '260' soit supérieur à 100 €.*




## g. OUTPUT

Cette clause permet de connaître les lignes affectées par l'opération du DML INSERT, UPDATE ou DELETE. Ce retour d'information permet à l'application qui a demandé l'exécution de l'instruction DML de savoir quelles lignes d'informations sont concernées.

La valeur des colonnes retournées par l'intermédiaire de la clause OUTPUT est celle après application des contraintes d'intégrité et exécution de l'instruction DML mais avant l'exécution du ou des déclencheurs associés à la table et à l'instruction DML.

---

Cette clause est disponible uniquement à partir de SQL Server 2005.

---

**Syntaxe**

```
OUTPUT [listeColonne] INTO @variable
```

`listeColonne`

Représente la liste des colonnes retournées dans la variable. Ces colonnes peuvent provenir directement de l'instruction DML, ou correspondre à un résultat de calcul élémentaire. Les données peuvent être issues directement de la table. En utilisant les préfixes DELETED et INSERTED, il est possible de voir les données touchées par la suppression/modification et après modification/insertion.

`@variable`

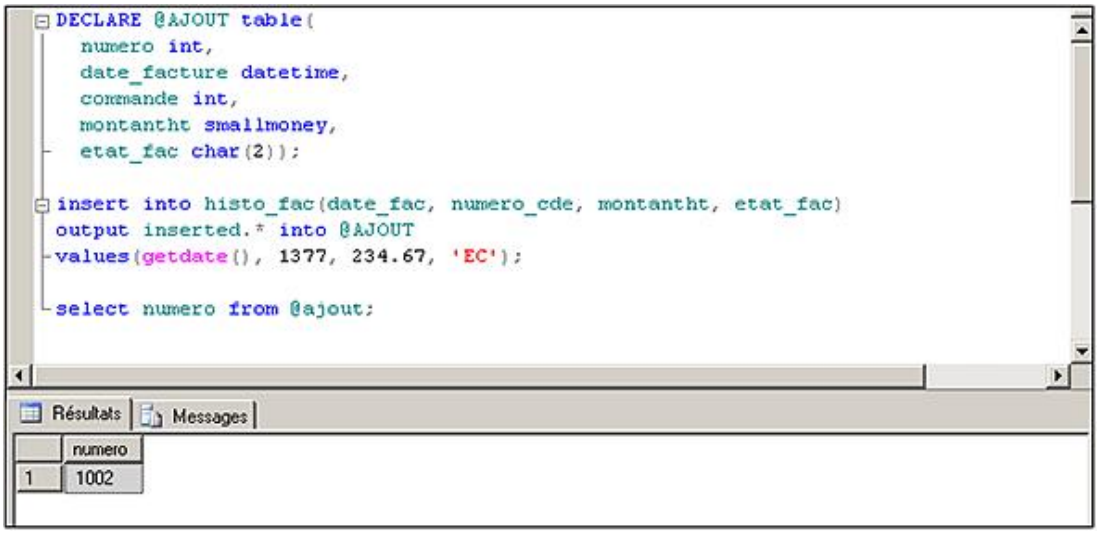
La variable doit être de type table et elle doit être déclarée avant son utilisation dans la clause OUTPUT.

```
INSERT INTO table(listeColonne)
OUTPUT  [listeColonne] INTO @variable
VALUES (listeValeurs)
```

```
DELETE table
OUTPUT[listeColonne] INTO @variable
WHERE condition

UPDATE table SET colonne=valeur
OUTPUT [listeColonne] INTO @variable
WHERE condition
```

*Exemple*

*Utilisation de la clause OUTPUT afin de connaître la valeur affectée par une colonne de type IDENTITY lors de l'ajout d'informations.*




## 6. Gestion des curseurs

L'utilisation de curseurs est une technique permettant de traiter ligne par ligne le résultat d'une requête, contrairement au SQL (SELECT) qui traite un ensemble de lignes.

Les curseurs peuvent être mis en œuvre par des instructions Transact SQL (curseurs ANSI-SQL) ou par l'API OLE-DB.

On utilisera les curseurs ANSI lorsqu'il faudra traiter les lignes individuellement dans un ensemble ou lorsque le SQL ne pourra pas agir uniquement sur les lignes concernées. Les curseurs des API seront utilisés par les applications clientes pour traiter des volumes importants ou pour gérer plusieurs ensembles de résultats.

---

N'utilisez les curseurs que si le SQL "ensembliste" n'est pas possible. Ils sont généralement coûteux en mémoire et en temps de traitement.

---

### a. DECLARE CURSOR

Cette instruction permet la déclaration et la description du curseur ANSI.

**Syntaxe**

```
DECLARE nomcurseur[INSENSITIVE][SCROLL]CURSOR
    FOR SELECT ....
    FOR {READ ONLY|UPDATE[OF liste_colonne]}]
```

`INSENSITIVE`

Seules les opérations sur la ligne suivante sont permises.

`SCROLL`

Les déplacements dans les lignes du curseur peuvent être effectués dans tous les sens.

UPDATE

Précise que des mises à jour vont être réalisées sur la table d'origine du curseur.

Un curseur INSENSITIVE avec une clause ORDER BY ne peut pas être mis à jour. Un curseur contenant ORDER BY, UNION, DISTINCT ou HAVING est INSENSITIVE et READ ONLY.

En plus de cette syntaxe conforme au standard ISO, Transact SQL propose une syntaxe étendue qui offre plus de possibilités au niveau des curseurs :

```
DECLARE nomcurseur CURSOR [ LOCAL | GLOBAL ]
     [ FORWARD_ONLY | SCROLL ]
     [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
     [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
     [ TYPE_WARNING ]
     FOR SELECT ...
     [ FOR UPDATE [ OF liste_colonne [ ,...n ] ] ]
```

LOCAL

La portée du curseur est locale au lot, procédure ou fonction en cours d'exécution dans lequel le curseur est défini. En dehors de ce lot, il n'est pas possible de référencer le curseur.

GLOBAL

La portée du curseur est globale à la connexion. L'option de base de données **default to local cursor** est définie à faux (false) par défaut. Cette option permet de définir la portée par défaut du curseur.

FORWARD_ONLY

Les données sont extraites du curseur dans leur ordre d'apparition (de la première à la dernière).

STATIC

Une copie temporaire des données est faite sur tempdb afin que le curseur ne soit pas affecté par les modifications qui peuvent intervenir sur la base.

KEYSET

Les lignes et leur ordre dans le curseur sont fixés au moment de l'ouverture du curseur. Les références vers chacune de ces lignes d'informations sont conservées dans une table temporaire au sein de tempdb.

DYNAMIC

Le curseur reflète exactement les données présentes dans la base. Ce qui signifie que le nombre de lignes, leur ordre et leur valeur peuvent être modifiés de façon dynamique.

FAST_FORWARD

Permet de définir un curseur en avant et en lecture seule (FORWARD_ONLY et READ_ONLY).

SCROLL_LOCKS

Permet de garantir le succès des instructions UPDATE et DELETE qui peuvent être exécutées relativement au curseur. Avec ce type de curseur, un verrou est posé lors de l'ouverture du curseur pour éviter qu'une autre transaction tente de modifier les données.

OPTIMISTIC

Avec cette option, il se peut qu'une opération de mise à jour (UPDATE) ou bien de suppression (DELETE) réalisée au sein du curseur ne puisse s'effectuer correctement car une autre transaction aura modifié les données en parallèle.

TYPE_WARNING

Un message d'avertissement (warning) est envoyé à l'application cliente si des conversions implicites de type sont effectuées.

### b. OPEN

Cette instruction permet de rendre le curseur utilisable et de créer éventuellement les tables temporaires associées. La variable @@CURSOR_ROWS est valorisée après le OPEN.

Compte tenu de l'espace disque et mémoire utilisé et du verrouillage éventuel des données lors de l'ouverture du curseur, cette opération doit être exécutée la plus proche possible du traitement des données issues du curseur.

**Syntaxe**

```
OPEN [GLOBAL] nomcurseur
```

### c. FETCH

C'est l'instruction qui permet d'extraire une ligne du curseur et de valoriser des variables avec leur contenu. Après le fetch, la variable @@FETCH_STATUS est à 0 si le fetch s'est bien passé.

**Syntaxe**

```
FETCH[{NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n]
     [FROM] [GLOBAL] nomcurseur[INTO Listevariable]
```

`NEXT`

Lit la ligne suivante (seule option possible pour les INSENSITIVE CURSOR).

`PRIOR`

Lit la ligne précédente.

`FIRST`

Lit la première ligne.

`LAST`

Lit la dernière ligne.

`ABSOLUTE n`

Lit la nième ligne de l'ensemble.

`RELATIVE n`

Lit la nième ligne à partir de la ligne courante.

### d. CLOSE

Fermeture du curseur et libération de la mémoire.

Cette opération doit intervenir dès que possible afin de libérer les ressources le plus tôt possible.

**Syntaxe**

```
CLOSE nomcurseur
```

### e. DEALLOCATE

Supprime le curseur et les structures associées.

**Syntaxe**

```
DEALLOCATE nomcurseur
```

*Exemple*

*On ne veut que les trois articles les plus chers par catégorie dans une table de travail :*

```
CREATE TABLE tempo_art(reference nchar(16), cat char(2), prix numeric(8,2));
DECLARE c_art CURSOR FOR
   SELECT reference_art, code_cat, prixht_art
     FROM articles ORDER BY code_cat, prixht_art DESC;
DECLARE @cpt int;
DECLARE @ref nchar(16);
DECLARE @cat char(2);
DECLARE @prix numeric(8,2);
DECLARE @catref char(2);
OPEN c_art; --ouverture du curseur
FETCH c_art INTO @ref, @cat, @prix; -- ramener la première ligne
WHILE (@@FETCH_STATUS=0) BEGIN-- Le dernier fetch a t il reùmené une ligne?
  INSERT INTO tempo_art VALUES(@ref, @cat, @prix);
  SELECT @catref=@cat; --mémoriser la catégorie
  SELECT @cpt=1;
  FETCH c_art INTO @ref, @cat, @prix; --ligne suivante
  WHILE (@@FETCH_STATUS=0 AND @catref=@cat) BEGIN
    IF (@cpt<3) BEGIN
      INSERT INTO tempo_art VALUES(@ref, @cat, @prix);
      SELECT @cpt=@cpt+1;
    END
    FETCH c_art INTO @ref, @cat, @prix; --ligne suivante
  END
END
CLOSE c_art;
DEALLOCATE c_art;
```

Si le même script Transact SQL travaille avec plusieurs curseurs, la variable @@FETCH_STATUS correspond à l'état du dernier curseur utilisé. Pour connaître avec détail l'état de chaque curseur, il faut interroger la colonne **fetch_status** de la vue dynamique **sys.dm_exec_cursors**.

# 7. Gestion des exceptions

## a. Les messages d'erreurs

Pour chaque erreur, SQL Server produit un message d'erreur. Ce message est, par défaut, affiché à l'écran et sa lecture complète permet bien souvent de résoudre le problème.

La plupart de ces messages sont définis dans SQL Server, mais il est également possible de définir ses propres messages par l'intermédiaire de la procédure **sp_addmessage**.

### La structure des messages

Quelle que soit leur origine, tous les messages d'erreurs possèdent la même structure et les mêmes champs d'informations qui sont :

- **numéro** : chaque message est identifié de façon unique par un numéro. Ce numéro est indépendant de la langue choisie pour installer SQL Server. Ainsi, le programme déclenche une erreur identifiée par son numéro. Ensuite, SQL Server sélectionne le message à afficher depuis la table **sys.messages** de la base master en fonction du numéro de l'erreur et de la langue d'installation du serveur.

- **message au format texte** : le message est spécifique à chaque message d'erreurs. Beaucoup de messages possèdent des variables afin d'adapter le message générique au cas précis et délivrer ainsi plus d'informations.

- **Sévérité** : c'est un indicateur sur la gravité de l'erreur. Ainsi des messages avec une sévérité de 1 ou 2 sont donnés simplement à titre informatif.

- **État** : une même erreur peut avoir différentes origines, c'est-à-dire être provoquée depuis différents contextes. Pour chaque type de contexte, un état est associé. Ce numéro d'état peut s'avérer utile lors de la recherche d'informations dans la documentation.

- **nom de la procédure** : si l'erreur est provoquée depuis une procédure stockée, alors son nom est affiché.

- **numéro de la ligne** : numéro de la ligne à l'origine de l'erreur, que cette ligne se trouve dans un lot d'instructions, une fonction, une procédure stockée ou bien un déclencheur de base de données.

### La gravité des messages d'erreurs

Dans SQL Server, chaque message d'erreur possède une gravité également nommée sévérité. Cette gravité permet de classer les messages par rapport au risque potentiel associé. Il existe 25 niveaux différents de gravité.

La gravité est représentée par un nombre entier compris entre 0 et 24. Cette sévérité est associée au message d'erreur lors de sa création, mais il est possible d'en fixer une légèrement différente lorsque l'erreur est levée par l'intermédiaire de l'instruction RAISERROR.

**Inférieure à 9**

Le message est donné à simple titre d'information. Il n'est pas bloquant. Si la gravité est de 0 alors le message n'est pas visible.

**Égale à 10**

C'est un message de type information pour décrire une erreur faite par l'utilisateur dans l'information qu'il vient de saisir.

**Entre 11 et 16**

L'erreur peut être résolue par l'utilisateur.

**Égale à 17**

Ressources insuffisantes. Signale, par exemple, que SQL Server manque d'espace disque, ou bien que l'instance a atteint certaines limites fixées par l'administrateur.

**Égale à 18**

Erreur interne non fatale. Une erreur interne s'est produite mais la requête a pu être traitée correctement et la connexion au serveur est maintenue.

**Égale à 19**

Erreur SQL dans l'accès à une ressource. SQL Server a atteint une limite non configurable de l'instance. À partir de ce niveau, tous les messages doivent être remontés à l'administrateur.

**Égale à 20**

Erreur fatale dans le processus courant. Le processus en cours vient de subir une erreur grave, mais l'instance SQL Server n'est pas affectée.

**Égale à 21**

Erreur fatale dans les processus de l'instance, cependant, il y a peu de chance que la base de données en elle-même soit affectée par ce problème.

**Égale à 22**

Erreur fatale concernant l'intégrité d'une table. L'intégrité des données contenues dans une table n'est pas respectée. L'administrateur peut utiliser DBCC CHECKDB pour savoir si d'autres tables sont affectées et tentera de rétablir l'intégrité par l'intermédiaire de DBCC.

**Égale à 23**

L'intégrité de la base n'est plus garantie. Si l'outil DBCC s'avère inefficace, l'administrateur peut être conduit à

restaurer la base pour remédier au problème.

**Égale à 24**

Problème matériel, tel qu'un échec d'accès au disque.

### b. Déclencher une erreur

Le programmeur peut décider de lever un message d'erreur en fonction du comportement du code. Pour lever une erreur, il va faire appel à l'instruction RAISERROR.

Il existe deux possibilités pour travailler avec cette instruction, soit elle lève une erreur parfaitement identifiée par son numéro, soit elle permet de lever une erreur prédéfinie.

**Syntaxe**

```
RAISERROR ( { identifiant | message }
    { ,gravité ,état }
    [ ,argument [ ,...n ] ] )
[WITH option, ...]
```

`identifiant`

Numéro d'identification du message tel qu'il est enregistré dans la table sysmessages. Le message sélectionné est fonction de son identifiant et de la langue de la session. Par défaut, c'est le message en langue anglaise qui est sélectionné.

`message`

Il est également possible de définir directement le texte du message, dans la limite de 2047 caractères. Dans ce cas, il est nécessaire de définir une gravité et un état.

`gravité`

Permet de spécifier la gravité du message d'erreur. En général les messages utilisateurs ont une gravité inférieure strictement à 20. Au-delà l'erreur est considérée comme fatale et la connexion à la base est rompue.

`état`

Permet de tracer l'origine de l'erreur.

`argument`

Dans le cas où le message d'erreur prédéfini ou non possède des paramètres, il faut les valoriser à partir de constantes ou bien de variables.

`option`

Il existe trois options possibles LOG, NOWAIT et SETERROR.

L'option LOG permet de spécifier que le message sera consigné dans l'observateur des évènements de Windows.

L'option NOWAIT permet de s'assurer que le message sera envoyé sans délai au client.

L'option SETERROR permet de valoriser @@ERROR et ERROR_NUMBER avec le numéro du message d'erreur.

*Exemples*

*Erreur définie par l'utilisateur :*

```
RAISERROR('!!Le stock est négatif!!',12,1);
```

Messages

```
Msg 50000, Niveau 12, État 1, Ligne 2
!!Le stock est négatif!!
```

*Erreur grave avec passage de paramètres dans le message :*

```
DECLARE @numero int;
SELECT @numero=1;
RAISERROR ('!!Erreur grave sur le client: %d !!',21,1,@numero)
     WITH LOG;
```

Messages

```
Msg 2745, Niveau 16, État 2, Ligne 4
Le processus ID 52 a provoqué l'erreur utilisateur 50000, gravité 21. SQL Serve
Msg 50000, Niveau 21, État 1, Ligne 4
!!Erreur grave sur le client: 1 !!
Msg 0, Niveau 20, État 0, Ligne 0
Une erreur grave s'est produite sur la commande actuelle. Les résultats éventue
```

### c. Définir un message d'erreur

Il est possible de définir ses propres messages d'erreur, afin de pouvoir faire remonter une erreur de type applicative vers l'application cliente en respectant le mécanisme de gestion des erreurs de SQL Server. En travaillant de cette manière au niveau des procédures stockées, des fonctions et des déclencheurs, il est très facile pour le programme appelant de gérer les erreurs.

Pour définir un nouveau message d'erreur, il faut faire appel à la procédure stockée **sp_addmessage**.

Les messages ajoutés à l'aide de la procédure **sp_addmessage** peuvent être supprimés par la procédure **sp_dropmessage**.

**Syntaxe**

```
sp_addmessage [ @msgnum = ] identifiant ,
     [ @severity = ] severité ,
     [ @msgtext = ] 'message'
     [ , [ @lang = ] 'langue' ]
     [ , [ @with_log = ] TRUE|FALSE ]
     [ , [ @replace = ] 'replace' ]
```

`@msgnum`

Nombre entier (int) qui permet de spécifier le numéro du message. Les messages définis par l'utilisateur doivent avoir un identifiant compris entre 50001 et 2 147 483 647.

`@severity`

Nombre entier (smallint) qui permet de préciser la gravité du message. Cette gravité doit être comprise entre 1 et 25.

`@msgtext`

Chaîne de caractères qui représente le texte affiché lorsque l'erreur est levée.

`@lang`

Chaîne de caractères qui permet de préciser la langue du message. En premier lieu, le message doit être défini en

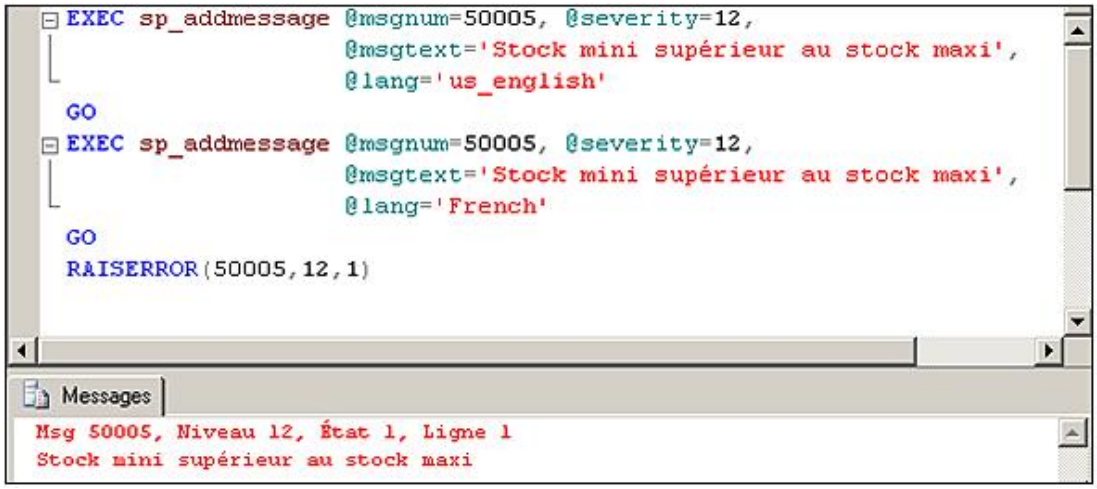
anglais 'us_english', puis il peut être localisé en français en précisant 'French'.

`@with_log`

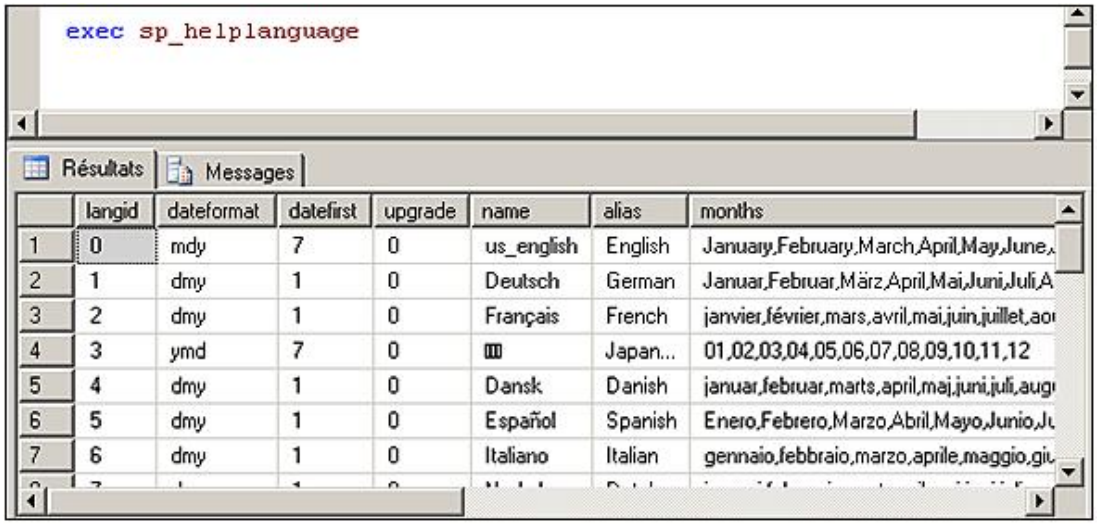
Permet de préciser si le message sera consigné dans l'observateur des évènements de Windows ou non.

`@replace`

Permet de demander d'écraser, s'il existe, le message qui porte le même numéro que celui en cours de création.




Pour connaître les différents codes de langue, il suffit d'exécuter la procédure `sp_helplanguage` sans aucun paramètre.




Il est possible de définir des variables dans le texte du message de façon à personnaliser le texte du message en fonction du contexte à partir duquel il est levé. Il est ainsi possible, par exemple, de connaître le nom de l'utilisateur, ou de la table affectée par l'opération.

Pour introduire des valeurs issues de variables dans le texte du message d'erreur il est nécessaire de préciser comment la valeur va être convertie et introduite dans le texte. Le format est toujours sous la forme suivante :

`%[drapeau][largeur][.precision]type.`

`drapeau`

Ce drapeau, qui peut être +, -, 0, # ou bien des espaces, a pour objectif le cadrage à gauche des données de type numérique avec complément éventuel avec des zéro et de préciser le comportement à adopter lors de l'affichage de valeurs signées.

`largeur`

Permet de définir le nombre de caractères que va réclamer la représentation textuelle de la valeur numérique.

`precision`

Permet de définir le nombre de décimales à afficher.

`type`

Les caractères suivants sont utilisés pour préciser le type de valeur transmis en paramètre :

**Type : Représente**

`d` ou `i` : Entier signé

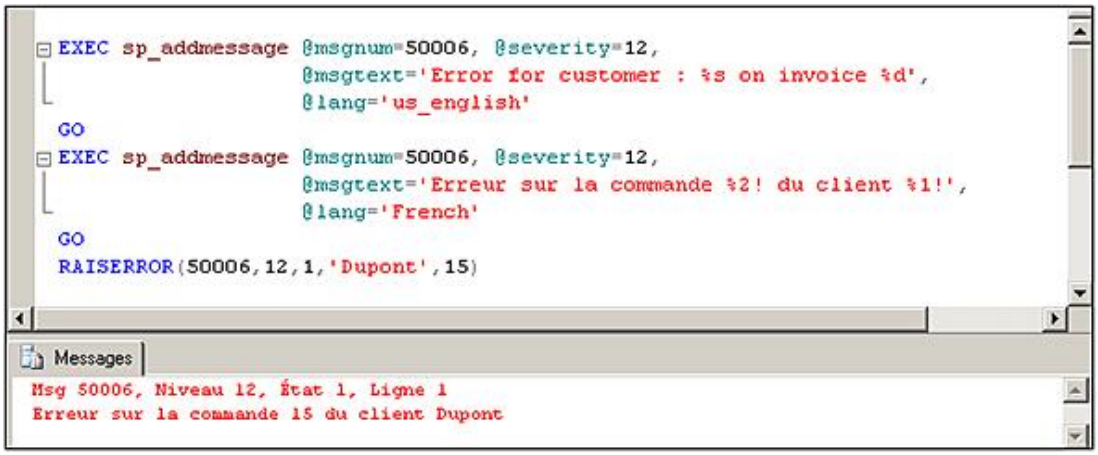
`u` : Entier non signé

`s` : Chaîne de caractères

`o` : Un nombre en octal (base 8)

`x` ou `X` : Un nombre en hexadécimal (base 16)

Lors de la définition du message en anglais, il faut utiliser les paramètres %s pour pouvoir insérer une donnée de type caractère et %d lorsque la valeur est de type numérique. Dans les versions localisées du message, il faut utiliser %1! pour faire référence au premier paramètre défini dans le texte du message anglais, puis %2! pour le deuxième et ainsi de suite.




Lorsque le message possède des paramètres, il est parfois préférable de préparer le message d'erreur en valorisant les différents paramètres du message lors de l'exécution du programme puis de lever l'erreur uniquement si cela est nécessaire.

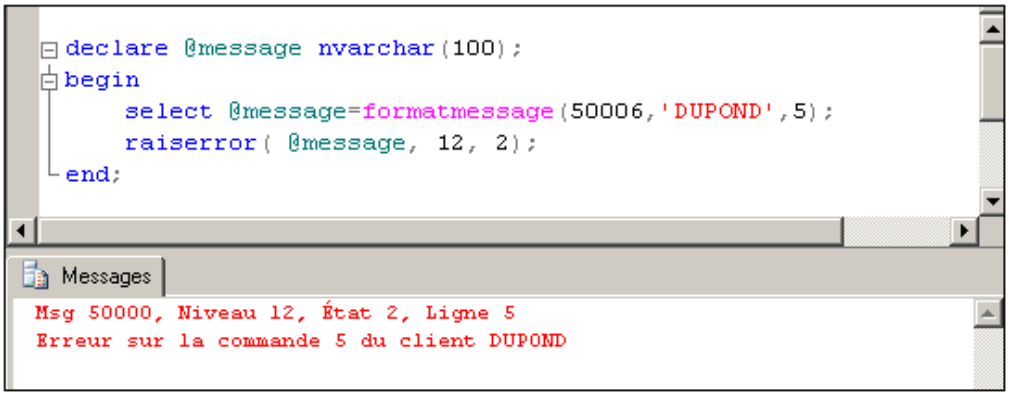
La fonction FORMATMESSAGE permet cette préparation du message. Le message ainsi préparé devra être levé par l'intermédiaire de la fonction RAISERROR. Cette possibilité de préparer ainsi les messages n'est possible que pour les messages qui possèdent des paramètres et qui ont été définis avec sp_addmessage.

**Syntaxe**

```
FORMATMESSAGE (idmessage, valeurParametre1,....)
```

*Exemple*

*Dans l'exemple suivant le message est préparé à l'aide de l'instruction FORMATMESSAGE :*




### d. La gestion des erreurs

En Transact SQL, il existe deux moyens de gérer les erreurs qui peuvent se produire lors de l'exécution du code.

- La première possibilité consiste à tester la valeur de la variable système @@error après chaque instruction pour savoir si elle s'est exécutée correctement ou non. Cette approche classique de la gestion des erreurs présente l'inconvénient d'être fastidieuse à écrire et de rendre le code peu lisible.

- La seconde possibilité consiste à regrouper une ou plusieurs instructions Transact SQL dans un bloc TRY et de centraliser la gestion des erreurs dans un bloc CATCH. Pour un type d'erreur, le traitement de l'erreur est écrit une seule fois même si plusieurs instructions peuvent lever cette même erreur. Ce type de gestion des erreurs est bien connu des développeurs Java, VC++, C#...

Dans le bloc CATCH, il est possible d'utiliser les fonctions ERROR_MESSAGE(), ERROR_NUMBER(), ERROR_SEVERITY(), ERROR_STATE() pour obtenir des informations sur le message d'erreur, le numéro de l'erreur, la gravité de l'erreur et l'état de l'erreur. En dehors de ce bloc, ces fonctions retournent systématiquement la valeur null et ne possèdent donc aucun intérêt.

Dans le cas où plusieurs blocs Try/Catch sont empilés, ces fonctions ne permettent d'obtenir des informations que sur l'erreur déclenchée localement et non pas dans l'un des sous-blocs.

Les blocs TRY et CATCH sont toujours associés. Il n'est pas possible de définir un bloc TRY sans bloc CATCH et réciproquement.

---

Il n'est possible de gérer dans le bloc CATCH que des erreurs dont la sévérité est supérieure à 10 et à condition que l'erreur ne mette pas fin au script.

---

**Syntaxe**

```
BEGIN TRY
...
END TRY
BEGIN CATCH
....
END CATCH;
```

#### Le bloc TRY

Le bloc TRY permet de regrouper l'ensemble des instructions qui sont susceptibles de lever des erreurs. Dans le cas où une instruction déclenche une erreur, le contrôle est immédiatement passé à la première instruction du bloc CATCH correspondant au bloc TRY.

Dans le cas où toutes les instructions du bloc TRY s'exécutent sans erreur, alors le bloc CATCH n'est jamais exécuté.

Dans un bloc TRY, il y a toujours au moins une instruction ou un bloc d'instructions, mais il peut y en avoir plusieurs.

Ce bloc est toujours entièrement défini à l'intérieur d'une procédure ou d'une fonction.

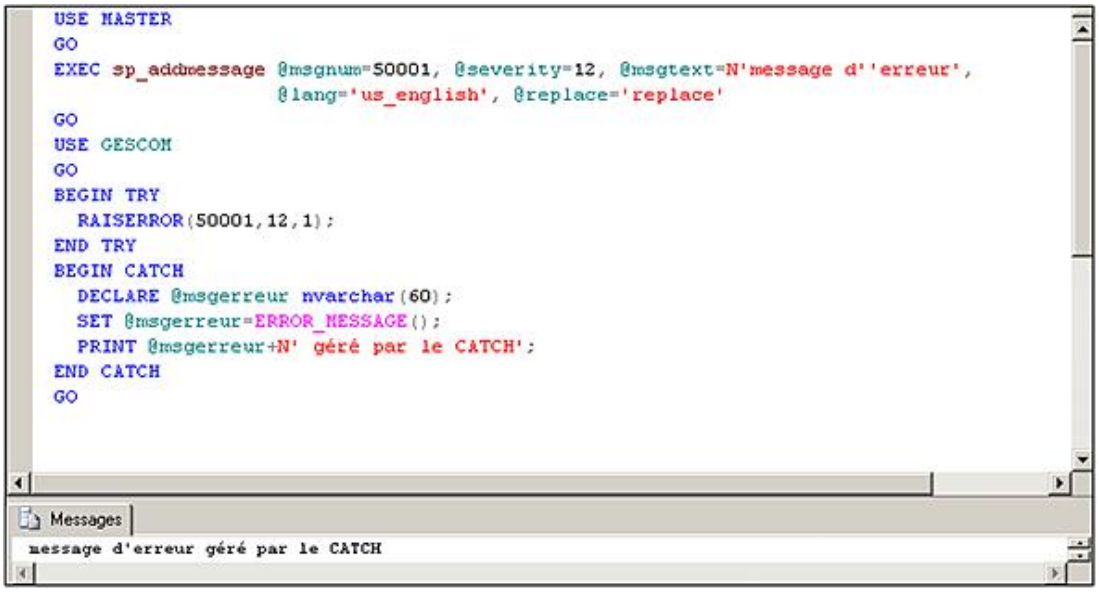
#### Le bloc CATCH

Ce bloc suit immédiatement le bloc TRY. Il est exécuté, si et seulement si, une instruction du bloc TRY a levé une erreur.

C'est dans ce bloc que du code est défini pour essayer de traiter au mieux les différents types d'erreurs qui peuvent avoir lieu au niveau du bloc TRY correspondant. Après le traitement de l'erreur, le contrôle est passé à l'instruction, qui suit immédiatement la fin du bloc CATCH (END CATCH).

Les différentes fonctions disponibles uniquement dans ce bloc sont : ERROR_MESSAGE(), ERROR_NUMBER(), ERROR_SEVERITY(), ERROR_ STATE(). Elles permettent d'obtenir toutes les informations relatives à l'erreur qui a provoqué l'exécution du bloc et d'adapter le traitement en fonction de cette erreur.

L'exemple suivant illustre la mise en place des blocs TRY et CATCH ainsi que la gestion d'une erreur dans le bloc CATCH.



# Gestion des procédures stockées

Les procédures stockées (*Stored Procedures*) sont des objets correspondant à un ensemble d'instructions du LMD, pouvant être exécutées par simple appel de leur nom ou par l'instruction EXECUTE. Ce sont de véritables programmes pouvant recevoir des paramètres, renvoyer des valeurs, être exécutés à distance, ayant leurs propres droits d'accès (privilège EXECUTE). De plus, les procédures stockées sont stockées dans le cache mémoire sous forme compilée lors de leur première exécution, ce qui accroît les performances (pour les exécutions suivantes !). Les procédures stockées peuvent être temporaires, c'est-à-dire créées pour une session (locale) ou plusieurs sessions (globale) du user.

Pour SQL Server une procédure stockée peut être définie comme une suite d'instructions Transact SQL, stockée dans la base de données et parfaitement identifiée par son nom. Pour permettre à cette suite d'instructions de s'adapter au plus grand nombre de cas, certaines valeurs du script sont paramétrables lors de l'appel de la procédure. Comme toute suite d'instructions Transact SQL, il est possible par exemple de trouver une instruction SELECT. L'exécution de la procédure déclenchera l'exécution de la requête et le résultat sera envoyé à l'environnement qui a demandé l'exécution de la procédure.

De nombreuses procédures stockées sont fournies par Microsoft et sont créées lors de l'installation des serveurs dans la base master. Ces procédures permettent de manipuler les tables système. Leur nom commence par "sp_".

Les différents cas d'utilisation de procédures stockées sont les suivants :

- Enchaînement d'instructions.
- Accroissement des performances.
- Sécurité d'exécution.
- Manipulation des données système.
- Mise en œuvre des règles d'entreprise.
- Traitements en cascade.

La création ou la modification des procédures stockées se fait par des instructions du Langage de Définition de données ou par SQL Server Management Studio.

**Syntaxe**

```
CREATE PROC[EDURE] nom[;numero][(param1[,...])][{FOR REPLICATION|WITH
RECOMPILE}][WITH ENCRYPTION]AS instructions.
```

`nom`

Nom d'objet unique dans la base. Précédé d'un signe #, la procédure sera temporaire locale, avec deux # elle sera temporaire globale.

`numéro`

Numéro d'ordre pour des procédures ayant le même nom.

`param1,...`

Paramètre sous la forme :

@nom type [ VARYING ][ = valeur ]

[ OUTPUT ], pouvant être passé à la procédure. OUTPUT permet de spécifier un paramètre retourné par la procédure.

VARYING spécifie le jeu de résultats pris en charge comme paramètre de sortie. S'applique uniquement aux paramètres de type cursor.

---

Une procédure stockée peut contenir au maximum 2100 paramètres.

FOR REPLICATION

Permet de préciser que la procédure sera utilisée lors de la réplication.

WITH RECOMPILE

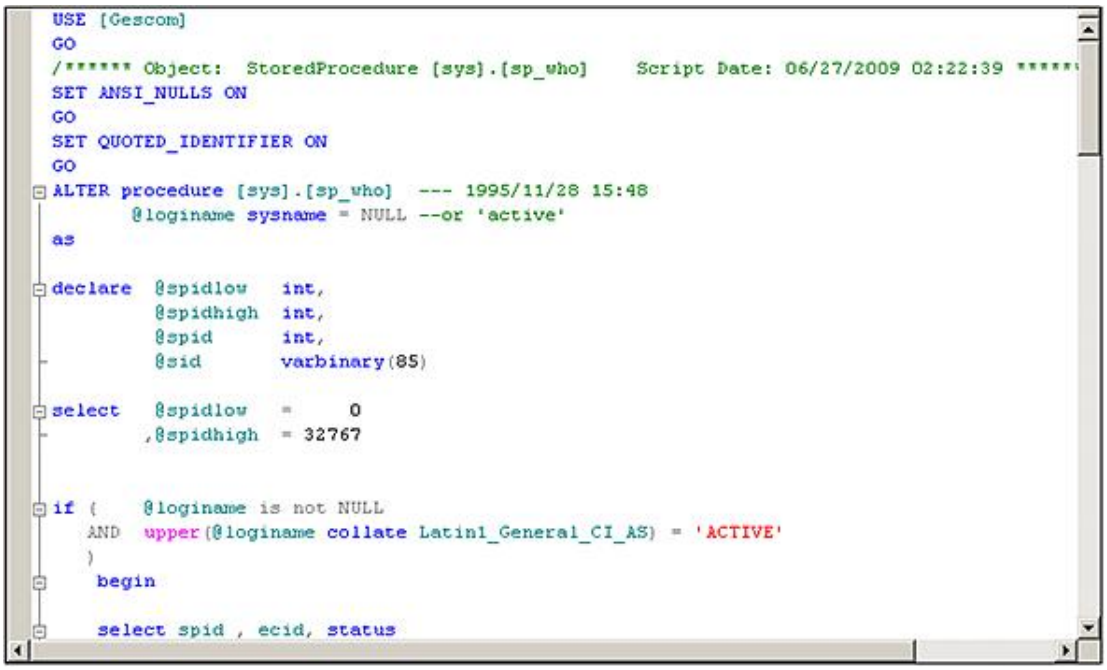
La procédure sera recompilée à chaque exécution.

WITH ENCRYPTION

Permet de crypter le code dans les tables système.

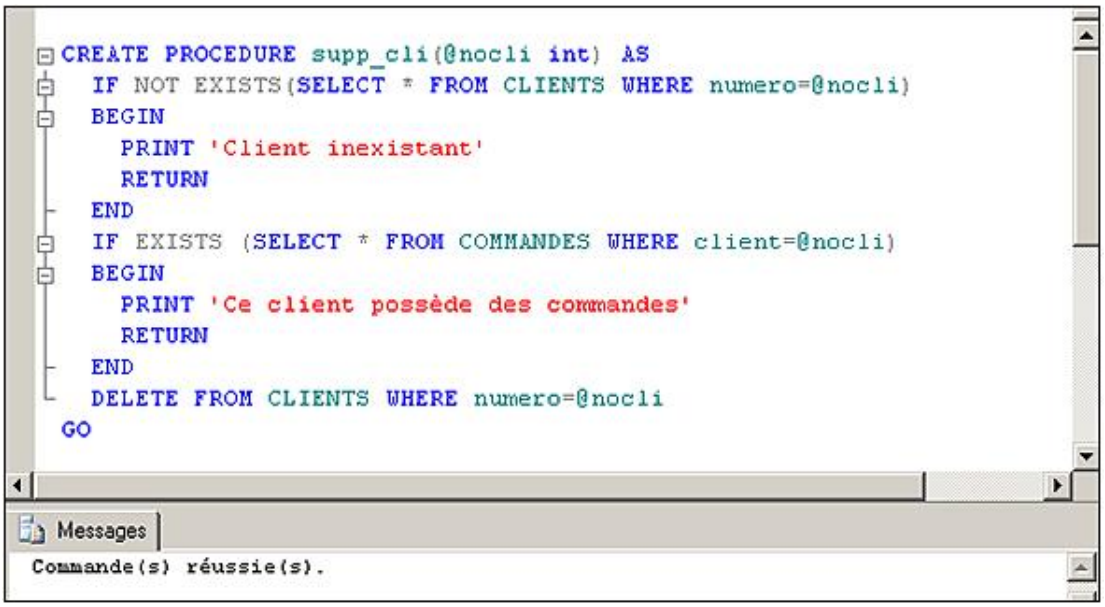
*Exemples*

*Code d'une procédure stockée système depuis Enterprise Manager :*



```
USE [Gescom]
GO
/****** Object:  StoredProcedure [sys].[sp_who]    Script Date: 06/27/2009 02:22:39 ******
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [sys].[sp_who]  --- 1995/11/28 15:48
        @loginame sysname = NULL --or 'active'
as

declare  @spidlow   int,
         @spidhigh  int,
         @spid      int,
         @sid       varbinary(85)

select   @spidlow   =     0
        ,@spidhigh  = 32767


if (    @loginame is not NULL
    AND  upper(@loginame collate Latin1_General_CI_AS) = 'ACTIVE'
    )
     begin

     select spid , ecid, status
```

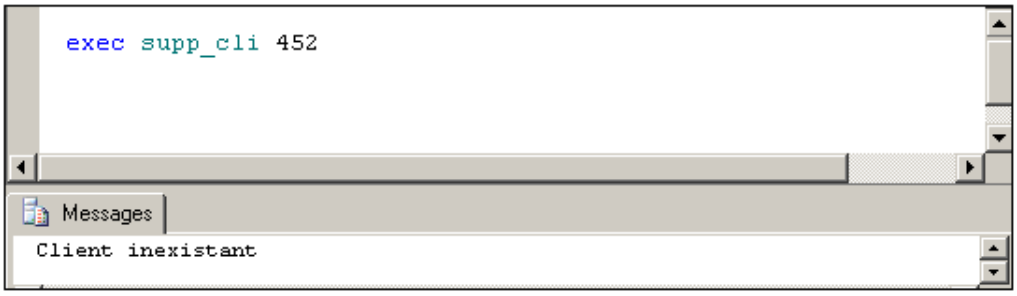
*Procédure utilisateur de suppression d'un client (création) :*



```
CREATE PROCEDURE supp_cli(@nocli int) AS
  IF NOT EXISTS(SELECT * FROM CLIENTS WHERE numero=@nocli)
  BEGIN
    PRINT 'Client inexistant'
    RETURN
  END
  IF EXISTS (SELECT * FROM COMMANDES WHERE client=@nocli)
  BEGIN
    PRINT 'Ce client possède des commandes'
    RETURN
  END
  DELETE FROM CLIENTS WHERE numero=@nocli
GO
```

```
Messages
Commande(s) réussie(s).
```

*Procédure utilisateur de suppression d'un client (utilisation) :*



# Les fonctions définies par l'utilisateur

SQL Server prend en charge trois types de fonctions utilisateurs : les fonctions scalaires, les fonctions tables en ligne et les fonctions tables multi-instructions.

Une fonction accepte entre 0 et 1024 paramètres d'entrée et retourne, soit une valeur scalaire, soit une table.

Pour chacun des paramètres de la fonction, il est possible de définir une valeur par défaut en utilisant le mot clé DEFAULT. Mais lors de l'appel de la fonction, il faut préciser le mot clé DEFAULT pour utiliser la valeur par défaut. Ce comportement est différent de celui des procédures où la valeur par défaut est prise en compte de façon automatique si le paramètre n'est pas précisé lors de l'appel.

Les fonctions de type scalaire retournent, à l'aide du mot clé RETURN, une valeur scalaire. Bien sûr, les données de type timestamp, d'un type de données défini par l'utilisateur et d'un type table ou cursor ne peuvent être renvoyées. Il en est de même pour les types de données text, ntext et image.

Les fonctions tables retournent comme résultat une table. Elles ne contiennent pas de corps et la table est le résultat d'une commande SELECT unique. Si la fonction est composée de plusieurs instructions, alors les instructions sont encadrées par les mots clés BEGIN et END.

Les fonctions disposent d'un champ d'action limité et elles ne peuvent en aucun cas modifier leur environnement d'exécution. Depuis une fonction, il n'est donc pas possible de modifier le contenu d'une table de la base de données. À l'intérieur d'une fonction, les seules actions possibles sont celles qui vont modifier les objets locaux à la fonction.

Dès que la fonction est créée à l'aide de l'instruction CREATE FUNCTION, la clause WITH SCHEMABINDING permet de lier la fonction à tous les objets auxquels elle fait référence. Dès lors, toute modification (**ALTER**) ou suppression (**DROP**) de ces objets est vouée à l'échec. Cette clause, non obligatoire, suppose que tous les objets référencés appartiennent à la même base de données et que le propriétaire de la fonction possède un droit de **REFERENCE** sur les objets référencés par la fonction.

## 1. Création d'une fonction

### Fonctions scalaires

```
CREATE FUNCTION nom_fonction ( [ liste_des_paramètres] )
RETURNS type_données
[ WITH ENCRYPTION| WITH SCHEMABINDING]
[ AS ]
BEGIN
corps de la fonction
RETURN valeur
END
```

### Fonctions tables en ligne

```
CREATE FUNCTION nom_fonction ( [ liste_des_paramètres] )
RETURNS TABLE
[ WITH ENCRYPTION| WITH SCHEMABINDING]
[ AS ]
RETURN [ (requête_SELECT ) ]
```

### Fonctions tables multi-instructions

```
CREATE FUNCTION nom_fonction ( [ liste_des_paramètres] )
RETURNS @@variable_retour TABLE (nom_colonne type, ...)
[ WITH ENCRYPTION| WITH SCHEMABINDING]
[ AS ]
BEGIN
corps de la fonction
RETURN valeur
END
```

*Exemples*

*Création de la fonction nbre_cde qui retourne le nombre de commandes passées par un client transmis en paramètre :*

```
CREATE FUNCTION nbre_cde(@nocli int)
   RETURNS int
AS
BEGIN
   DECLARE @nbre int
   SELECT @nbre=count(*)
     FROM COMMANDES
    WHERE client=@nocli
   RETURN @nbre
END
GO
```

```
Messages
Commande(s) réussie(s).
```

*Création d'une fonction table inline qui permet de connaître les articles qui possèdent un prix inférieur à celui passé en paramètre :*

```
CREATE FUNCTION ArticlesPetitPrix (@seuil int)
   RETURNS TABLE
AS
   RETURN (SELECT * FROM ARTICLES WHERE prixht_art<@seuil)
```

```
Messages
Commande(s) réussie(s).
```

*Création d'une fonction qui retourne une table et qui permet de connaître, pour le client passé en paramètre, le nombre de commandes passées et le montant moyen de ces commandes.*

```
CREATE FUNCTION AnalyseClient(@nocli int)
  RETURNS @FicheClient TABLE (libelle nchar(30), valeur int)
AS
BEGIN
  INSERT INTO @FicheClient
    VALUES('Nombre de commandes',dbo.nbre_cde(@nocli))
  INSERT INTO @FicheClient
    SELECT 'Montant total', convert(int, SUM(quantite*prixht_art))
      FROM COMMANDES cde INNER JOIN LIGNES_CDE lig
        ON cde.numero=lig.commande
     INNER JOIN ARTICLES art
        ON lig.article=art.reference_art
  RETURN
END
```

```
Messages
Commande(s) réussie(s).
```

*Utilisation de la fonction nbre_cde :*

```
SELECT dbo.nbre_cde(160001)
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | 1 |

*Utilisation de la fonction de type table :*

```
SELECT * FROM ArticlesPetitPrix(500);
```

Résultats | Messages

| | REFERENCE_ART | DESIGNATION_ART | PRIXHT_ART | code_cat |
|---|---|---|---|---|
| 1 | 0001C001 | Papier photo autocollant 10x15 - PS-101 | 6.05 | 231 |
| 2 | 000387 | SpeedÆCom V92 Ready V3 | 57.89 | 236 |
| 3 | 000393 | PCI 56000 V92 Ready V4 | 13.89 | 232 |
| 4 | 000397 | USB ADSL v4 | 49.90 | 235 |

*Utilisation de la fonction de type table multiligne :*

```
SELECT * FROM AnalyseClient(160001)
```

Résultats | Messages

| | libelle | valeur |
|---|---|---|
| 1 | Nombre de commandes | 1 |
| 2 | Montant total | 2922554 |

*Création d'une fonction avec l'option WITH SCHEMABINDING et tentative de suppression de la table utilisée par la fonction :*

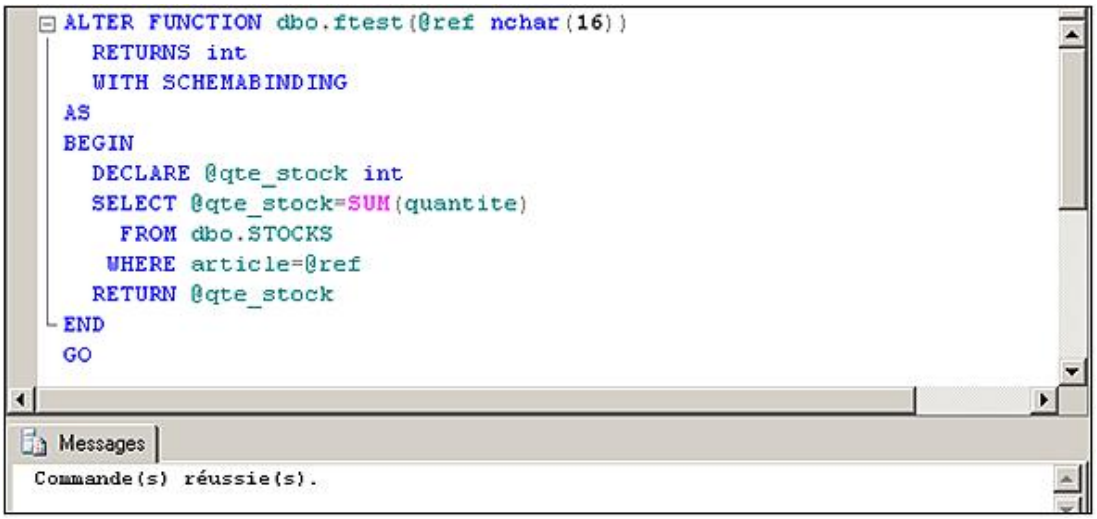
```
CREATE FUNCTION ftest()
  RETURNS int
  WITH SCHEMABINDING
AS
BEGIN
  DECLARE @retour int
  SELECT @retour=count(*) FROM dbo.HISTO_FAC
  RETURN @retour
END
go
DROP TABLE HISTO_FAC
```

Messages

```
Msg 3729, Niveau 16, État 1, Ligne 1
Impossible de DROP TABLE 'HISTO_FAC' car il est référencé par l'objet 'ftest'.
```

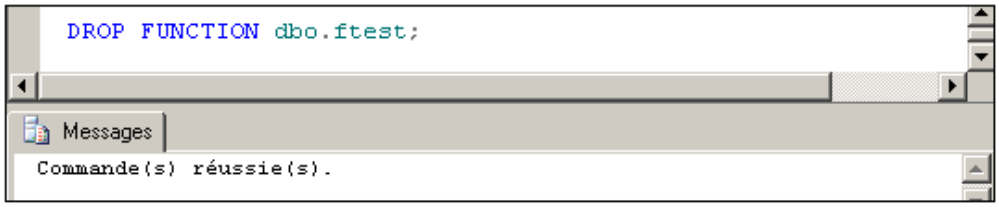
## 2. Modification d'une fonction

La commande ALTER FUNCTION accepte les mêmes paramètres que CREATE FUNCTION. La différence principale réside dans le fait que la commande ALTER FUNCTION permet de réécrire le corps d'une fonction qui existe déjà dans la base tandis que la commande CREATE permet de créer une nouvelle fonction. La commande ALTER ne permet pas, par exemple, de changer simplement une ligne de code mais elle permet de changer la totalité de la définition de la fonction.




## 3. Suppression d'une fonction

C'est la commande DROP FUNCTION qui permet de supprimer une fonction.




Les procédures **stockées sp_help** et **sp_helptext** permettent d'obtenir plus d'informations sur les fonctions définies par l'utilisateur.

## 4. CROSS APPLY et OUTER APPLY

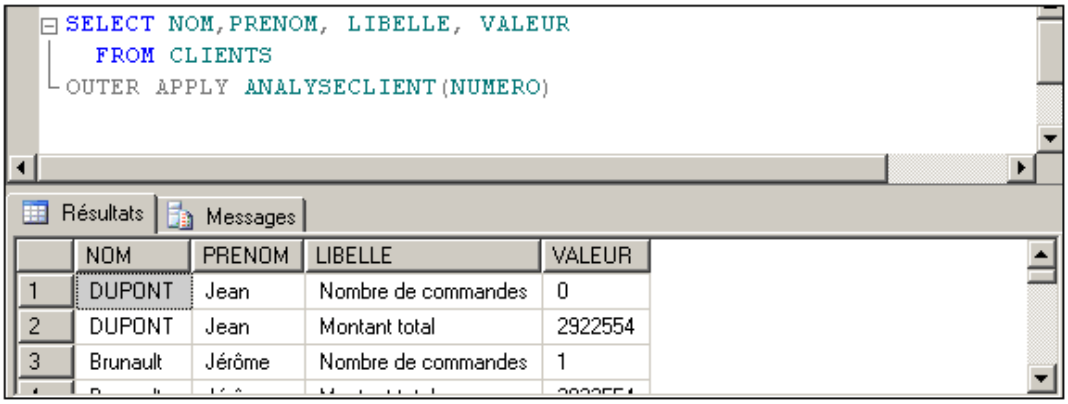
L'opérateur Xxxxx APPLY permet de mettre en relation les données issues d'une table/vue de la base de données et une fonction de type table.

Avec CROSS APPLY, toutes les lignes de la table/vue de la base de données sont présentes dans le jeu de résultats même si la fonction de type table retourne une valeur nulle.

Avec OUTER APPLY, seules les lignes de la table/vue pour lesquelles la fonction table retourne une valeur non nulle sont présentes dans le jeu de résultats.

*Exemple*

*Mettre en relation les informations des clients (nom et prénom) en relation avec l'analyse qui est faite de leurs commandes par la fonction AnalyseClient qui retourne une table.*



# Le contexte d'exécution

Le contexte d'exécution est directement lié à la connexion et à l'utilisateur de base de données associé. Le contexte d'exécution permet d'établir la liste des actions possibles et celles qui ne le sont pas. Cette liste est établie à partir des privilèges accordés à l'utilisateur soit directement soit par l'intermédiaire de rôles.

Dans certains cas, il peut être nécessaire et souhaitable de modifier le contexte d'exécution afin de profiter de privilèges étendus mais uniquement dans le cadre d'un script, d'une procédure ou d'une fonction.

## EXECUTE AS

À l'aide de cette instruction il est possible de demander à se connecter sur la base en utilisant une connexion différente de celle en cours. Cette instruction peut être exécutée de façon autonome dans un script Transact SQL ou bien être utilisée en tant que clause lors de la création d'une procédure, fonction ou trigger.

Pour les procédures, fonctions et trigger la clause EXECUTE AS donne beaucoup de souplesse en terme de programmation et permet à un utilisateur de réaliser des actions pour lesquelles il ne possède pas de privilèges. Pour le développeur, le changement de contexte d'exécution permet également de garantir que la bonne exécution du code ne sera pas entravée pas un problème de droits d'accès aux données.

Lors de l'exécution d'un script Transact SQL, l'instruction EXECUTE AS permet de réaliser par exemple de façon ponctuelle des opérations qui nécessitent des privilèges élevés alors que le reste du script ne le nécessite pas.

## SETUSER

Contrairement à l'instruction EXECUTE AS qui ne modifie pas la connexion initiale au serveur, l'instruction SETUSER permet de changer de connexion au cours d'un script Transact SQL. C'est-à-dire que le contexte actuel d'exécution est clôturé et un nouveau contexte d'exécution est ouvert. Il n'est pas possible de revenir au premier contexte d'exécution.

## Original_Login

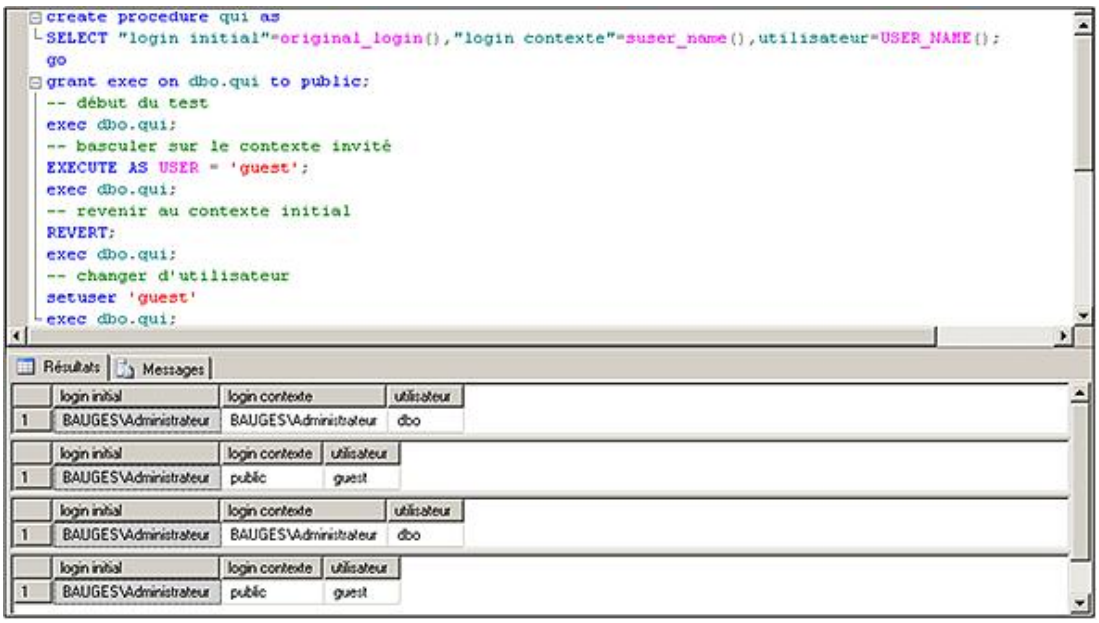
Cette fonction permet de déterminer le nom exact de la connexion utilisée initialement pour se connecter au serveur. La connaissance de ce nom ne présente un intérêt que lorsque que le contexte d'exécution diffère de la connexion initiale. Le contexte d'exécution peut être modifié par l'intermédiaire de l'instruction EXECUTE AS.

## REVERT

Suite au changement de contexte d'exécution à l'aide de l'instruction EXECUTE AS, l'instruction REVERT permet de revenir au contexte d'exécution présent lors du changement de contexte avec EXECUTE AS.

*Exemple*

*Dans l'exemple présenté ci-dessous, la procédure dbo, qui est définie afin d'afficher toutes les informations relatives à la connexion initiale au serveur, la connexion utilisée pour exécuter la requête et enfin le compte d'utilisateur utilisé :*



# Les déclencheurs

SQL Server propose deux types de déclencheurs : les déclencheurs du DML et ceux du DDL.

Les déclencheurs DML existent depuis longtemps dans SQL Server et sont présents dans de nombreuses bases de données. C'est ce type de déclencheur qui est détaillé ici.

Les déclencheurs du DDL repose sur le même principe, à savoir associer l'exécution d'une procédure stockée à l'exécution d'une instruction. La particularité tient ici du fait que le déclencheur va être associé à une instruction du DDL soit une commande CREATE, ALTER DROP, GRANT, DENY, REVOKE et UPDATE STATISTICS. L'objectif de ces déclencheurs est de suivre l'évolution de la base pour réaliser au mieux les différentes tâches administratives.

## Syntaxe

```
CREATE TRIGGER nom_trigger ON { table | vue }
 [ WITH ENCRYPTION ]
 {FOR | AFTER | INSTEAD OF } {  INSERT , UPDATE ,DELETE }
  [ WITH APPEND ]   [ NOT FOR REPLICATION ]
  AS
  [ IF UPDATE ( colonne )
  | IF ( COLUMNS_UPDATED ( )opérateur_comparaison_bits)]
Instructions_SQL
```

`WITH ENCRYPTION`

La définition du déclencheur est enregistrée de façon cryptée. Il n'est donc pas possible de connaître le code du déclencheur a posteriori. Cette option évite également que le déclencheur soit publié dans le cadre d'une réplication.

`FOR`

Permet de préciser à quel ordre SQL DML le déclencheur est associé. Par défaut, le déclencheur est de type AFTER.

`AFTER`

C'est le mode par défaut des déclencheurs. Le code est exécuté après vérification des contraintes d'intégrité et après modification des données.

`INSTEAD OF`

Le corps du déclencheur est exécuté à la place de l'ordre SQL envoyé sur la table ou la vue. Ce type de déclencheur est particulièrement bien adapté pour les vues.

`INSERT, UPDATE, DELETE`

Un déclencheur peut agir par rapport à une ou plusieurs actions. Dans ce cas, on séparera les actions par des virgules.

`WITH APPEND`

Cette clause n'est nécessaire que si le niveau de compatibilité de la base est inférieur ou égal à 65. Elle permet alors d'ajouter plusieurs déclencheurs sur un même ordre SQL et un même objet. Ce comportement est celui par défaut depuis la version (70).

`NOT FOR REPLICATION`

Indique que le déclencheur ne doit pas être exécuté lorsque la modification des données est issue d'un processus de réplication.

`IF UPDATE (colonne)`

Ne peut être utilisé que pour les déclencheurs UPDATE ou INSERT et ne s'exécutera que si la ou les colonnes sont concernées.

`IF ( COLUMNS_UPDATED ( ) opérateur_comparaison_bits)`

Cette fonction permet de connaître les indices de la ou des colonnes qui ont été mises à jour. Pour chaque colonne affectée par la mise à jour, un bit est levé. Pour savoir quelles ont été les colonnes mises à jour, une simple comparaison binaire suffit.

*Instructions_SQL*

Il est possible d'utiliser toutes les instructions Transact SQL de manipulations de données (DML). Les instructions suivantes ne sont pas autorisées :

- CREATE et DROP.
- ALTER TABLE et ALTER DATABASE.
- TRUNCATE.
- GRANT et REVOKE.
- UPDATE STATISTICS.
- RECONFIGURE.
- LOAD DATABASE.

Il possible de définir, sur une même table, plusieurs déclencheurs pour chaque opération INSERT, UPDATE et DELETE

Les déclencheurs sont exécutés après (AFTER) vérification des contraintes d'intégrité et insertion des données dans la table.

Si l'instruction SQL échoue, alors le déclencheur n'est pas exécuté.

La procédure stockée **sp_helptrigger** permet de connaître les déclencheurs définis sur une table.

Les déclencheurs de type AFTER peuvent être posés uniquement sur les tables, il n'est pas possible de poser de tels déclencheurs sur les vues. Il est possible de créer plusieurs déclencheurs AFTER pour une même table et un même ordre SQL (insert, update ou delete) de déclenchement. Si plusieurs déclencheurs existent, la procédure stockée **sp_settriggerorder** permet de fixer le déclencheur qui s'exécutera en premier et celui qui s'exécutera en dernier. Les autres déclencheurs s'exécuteront suivant un ordre non maîtrisable.

Si une table possède une action en cascade, il n'est plus possible de poser un déclencheur INSTEAD OF.

SQL Server peut autoriser la récursivité des triggers si l'option de base de données recursive triggers a été activée à l'aide de ALTER DATABASE. Cette option permet de donner une puissance nettement supérieure aux déclencheurs mais elle peut être parfois dangereuse à utiliser.

Parmi les récursions possibles, il faut distinguer la récursion directe de la récursion indirecte.

La récursion directe est gérée par le paramètre **recursive_triggers**. Elle autorise, par exemple, un déclencheur posé sur la table CLIENTS et associé à l'ordre INSERT de contenir un ordre INSERT sur cette même table CLIENTS. Ce second ordre INSERT déclenche lui aussi l'exécution du déclencheur.

La récursion indirecte est gérée par le paramètre de serveur **nested triggers** de la procédure **sp_configure**. Pour illustrer la récursion indirecte, il faut considérer qu'un déclencheur posé sur la table de COMMANDES et associé à l'ordre INSERT provoque un ordre INSERT dans la table des CLIENTS, et que le déclencheur associé provoque à son tour un ordre INSERT sur la table des COMMANDES.

Afin d'éviter tout blocage définitif, les déclencheurs ne peuvent pas compter plus de 32 niveaux d'imbrication.

Lors des modifications de données, SQL Server crée des lignes dans des tables de travail ayant la même structure que la table modifiée : les tables **inserted** et **deleted**.

Lors d'une commande INSERT, la table **inserted** contient une copie logique des lignes créées.

Lors d'une commande DELETE, les lignes supprimées sont placées dans la table **deleted**.

Lors d'une commande UPDATE, les lignes contenant les modifications sont placées dans la table **inserted**, les lignes à modifier dans la table **deleted**.

Les tables **inserted** et **deleted** peuvent être accessibles pendant l'exécution du trigger.

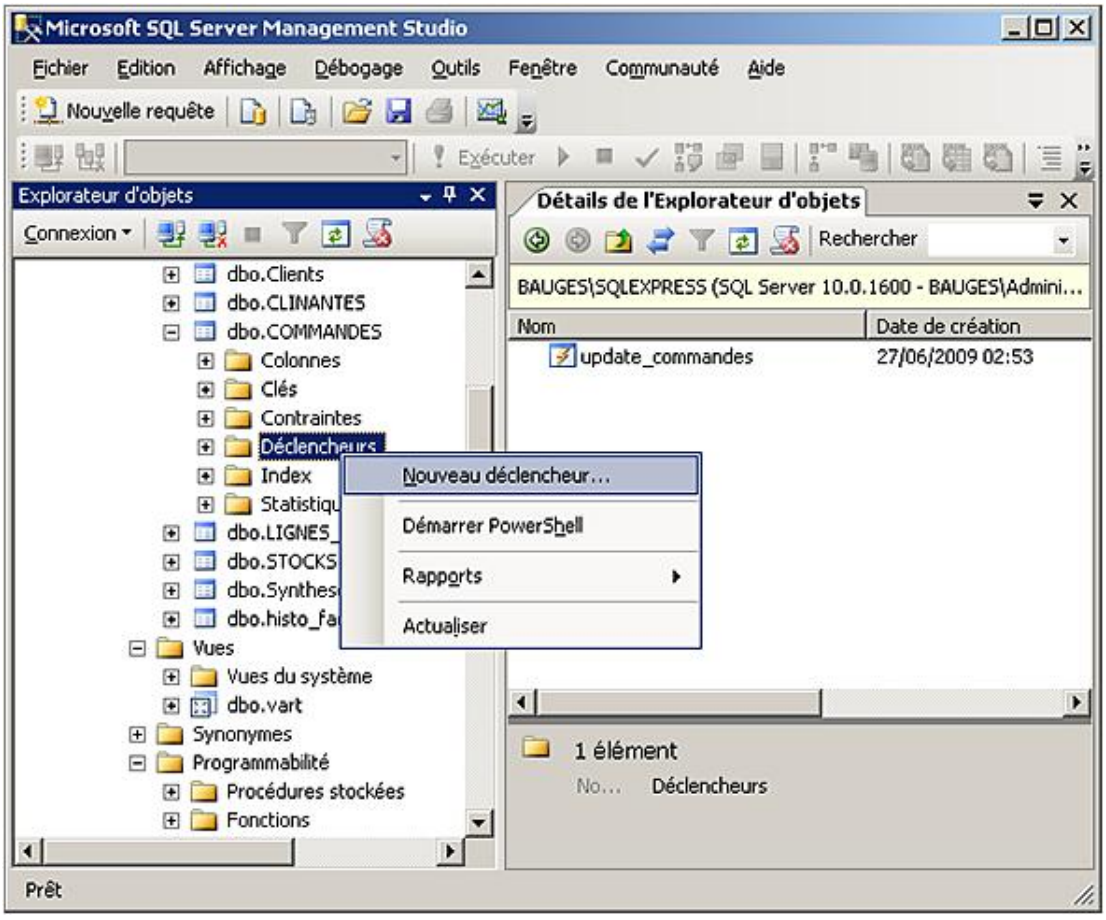
*Exemples*

*Création automatique d'une facture lors du changement d'état de la commande :*

```
create trigger update_commandes on commandes
after update
as
if update (etat)
  begin
    declare @etat as char(2)
    declare @numero as int
    select @etat=etat, @numero=numero
      from inserted
    if (@etat='LI')
      insert into histo_fac(numero_cde, date_fac)
        values(@numero, getdate())
  end;
```

```
Messages
Commande(s) réussie(s).
```

*Demande de création d'un déclencheur sur la table Commandes depuis SQL Server Management Studio :*




*Création d'un premier déclencheur sur la table Commandes :*

```
create trigger ins_cde_date on commandes
AFTER insert
as
BEGIN

  SET NOCOUNT ON;
  declare @numero int
  select @numero=numero from inserted
  -- la date de commande est la date du jur
  update commande set date_cde=getdate() where numero=@numero
END;
```

Messages

```
Commande(s) réussie(s).
```

*Création d'un second déclencheur sur la table Commandes pour calculer le taux de remise :*

```
create trigger ins_cde_taux on commandes
AFTER insert
as
BEGIN
  SET NOCOUNT ON
  declare @numero int
  declare @nombre_commandes int
  select @numero=numero from inserted
  -- compter le nombre de commandes pour ce client
  select @nombre_commandes=count(*)
  from commandes, inserted
  where commandes.client=inserted.client
  -- remise de 5% à partir de 10 commandes
  if @nombre_commandes>10
    update commandes set taux_remise=5 where numero=@numero
end;
```

Messages

```
Commande(s) réussie(s).
```

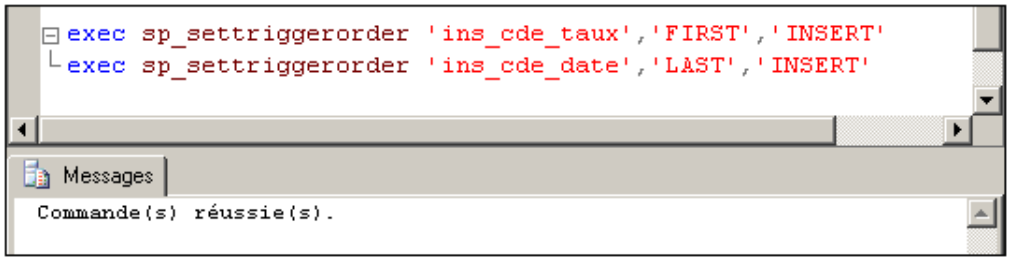
*Connaître les déclencheurs posés sur la table des commandes :*

```
exec sp_helptrigger 'dbo.commandes';
```

Résultats | Messages

| | trigger_name | trigger_owner | isupdate | isdelete | isinsert | isafter | isinsteadof | trigger_schema |
|---|---|---|---|---|---|---|---|---|
| 1 | update_commandes | dbo | 1 | 0 | 0 | 1 | 0 | dbo |
| 2 | ins_cde_date | dbo | 0 | 0 | 1 | 1 | 0 | dbo |
| 3 | ins_cde_taux | dbo | 0 | 0 | 1 | 1 | 0 | dbo |

*Fixer le premier et le dernier déclencheur qui s'exécutent lors de chaque opération d'INSERT sur la table des commandes :*



# Introduction

Les données utilisées dans le contexte des applications ne cessent d'évoluer. Il est donc normal que les serveurs de bases de données évoluent également en proposant des types adaptés à ces nouveaux formats. C'est ce que fait SQL Server en offrant la possibilité de stocker des données au format xml, des données géographiques ainsi qu'une meilleure gestion des documents annexes (image, vidéo, son, document numérisé...) afin de ne pas alourdir le processus de gestion de la base tout en liant les données relationnelles à ces informations stockées directement sur le système de fichiers.

# Travailler avec le format XML

Les données au format XML sont de plus en plus présentes dans l'environnement de travail. Il est donc normal qu'une base de données s'adapte afin d'être en mesure de stocker et de gérer de façon optimale les données définies dans ce format. C'est ce que fait SQL Server en offrant la possibilité de travailler directement avec des données au format XML et de les stocker dans la structure relationnelle d'une table. Étant donné que XML représente avant tout un format d'échange de données, SQL Server propose également les outils nécessaires pour produire un document XML à partir de données relationnelles ou bien au contraire d'intégrer dans les tables relationnelles des données issues d'un document XML.

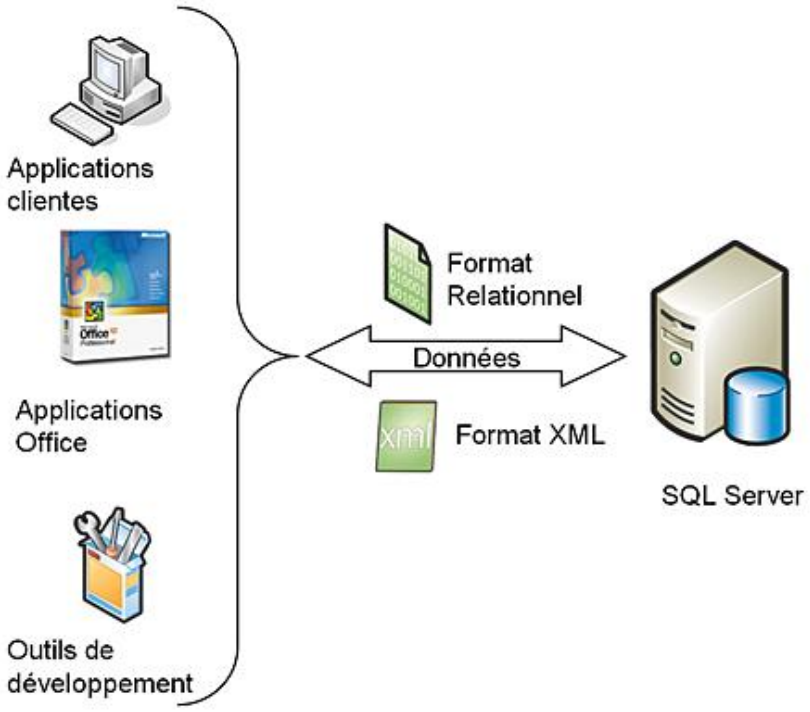
Il est possible de stocker les informations, soit au format relationnel, soit au format XML. Chaque format possède ses avantages et ses inconvénients.

SQL Server héberge un moteur relationnel pour le stockage et le travail avec les données conservées à ce format. Mais SQL Server propose également de gérer des données au format XML. Ainsi, quel que soit le mode de stockage retenu, SQL Server peut héberger ces données dans leur format natif.

L'objectif de SQL Server est d'être capable de s'adapter au mode de stockage des données en fonction du format avec lequel travaille l'application cliente.

Microsoft Office 2007 permet aux utilisateurs de Word, Excel Visio et Infopath de générer leurs documents au format XML, par l'intermédiaire du format OpenXML.

Le schéma ci-dessous illustre le fait que les applications travaillent aussi bien avec des données au format relationnel qu'au format XML.




Pour répondre correctement aux différentes demandes, SQL Server a considérablement amélioré sa gestion du format XML.

## Choisir un format

Les deux formats ne sont pas en concurrence mais sont complémentaires. Le moteur de base de données doit donc être capable de gérer de façon optimum les données, quel que soit leur format de stockage.

Le XML est particulièrement bien adapté pour l'échange d'informations entre des applications, pour la gestion documentaire, pour l'envoi de message (SOAP)...

Le XML présente l'avantage d'être autodescriptif. Il offre ainsi la possibilité de transférer des structures de données complexes. Cette représentation des données est faite sous forme d'arborescence.

Cependant, le format relationnel permet de garantir une meilleure homogénéité des données car les tables sont fortement structurées. La structuration tabulaire des données donne la possibilité de stocker un grand volume d'informations de façon fiable et les requêtes pour extraire ces données sont rapides et performantes. C'est sans aucun

doute le meilleur format pour stocker et travailler avec de gros volumes d'informations.

Le tableau suivant permet de privilégier un format ou un autre en fonction de la structure initiale des données :

| Formatage des données | XML | Relationnel |
|---|---|---|
| Fichier plat | Bien adapté | Bien adapté |
| Structure hiérarchique | Bien adapté | Possible |
| Données semi structurées | Bien adapté | Possible |
| Langage de description | Bien adapté | Possible |
| Conserver l'ordre | Bien adapté | Possible |
| Récursivité | Bien adapté | Possible |

## 1. Le type XML

SQL Server propose un type de données XML pour stocker les données au format natif XML. Ce type n'est pas un champ texte de grande dimension, nvarchar(max) par exemple. En effet, si pour des raisons de stockage pur cela ne change pas grand-chose, le type XML permet de faire des recherches précises d'informations. Il est également possible de définir des index sur des colonnes de type XML afin d'accélérer le traitement des requêtes.

Il est possible de créer des tables relationnelles qui, en plus des colonnes de types habituels, peuvent compter une ou plusieurs colonnes de type XML.

Les colonnes de type XML utilisent un format binaire (blob) pour stocker l'information dans la base relationnelle, ainsi le document XML est conservé dans l'état. De fait, l'espace pour chaque donnée XML est limité à 2 Go. De plus, le document XML ne doit pas être structuré avec une hiérarchie contenant plus de 128 niveaux.

Les données XML sont typées en UTF-16 par SQL Server.

Avec le type de données XML, pour stocker les données directement à ce format, SQL Server évite de concevoir un travail long et fastidieux de mappage entre le format XML et la structure relationnelle des données telles qu'elles sont organisées dans la base. Ce type dispose des méthodes query(), exist(), value(), nodes() et modify() afin de pouvoir travailler sur les données de façon simple. Ces méthodes s'appuient sur XQuery, sous ensemble de XML spécifique aux requêtes.

Afin de satisfaire aux exigences du W3C, il est possible d'associer une collection de schémas à une colonne de type XML. Les données stockées dans la colonne devront alors respecter les contraintes du schéma. Cette colonne sera alors dite XML typé, sinon il s'agit d'une colonne XML non typé.

### XML non typé

Le type XML, tel qu'il est défini dans SQL Server, respecte la définition donnée par le standard ISO SQL-2003. À ce titre, il se doit d'accueillir des documents XML 1.0 bien formés ainsi que des fragments XML.

La colonne qui utilise un type XML non typé, c'est-à-dire non relié à un schéma XML, ne contiendra toutefois que des informations conformes à un document XML 1.0 bien formé, ou bien un fragment XML.

Cette méthode de fonctionnement est certainement la plus souple, cependant, lorsque l'on dispose d'un schéma XML, il est préférable de passer par un type XML typé.

Elle doit être utilisée lorsque l'on ne dispose pas de schéma XML, ou lorsqu'il existe plusieurs schémas XML avec des liaisons à des sources de données externes.

*Exemple*

*La table catalogue est créée avec une colonne de type XML non typé :*

```
CREATE TABLE CATALOGUE(
   numero int CONSTRAINT PK_CATALOGUE PRIMARY KEY,
   page xml);
```

```
Messages
Commande(s) réussie(s).
```

## XML typé

Dans le cas où les informations qui vont être contenues dans une colonne de type XML, sont décrites dans une collection de schémas XML, il est possible d'associer cette collection de schémas XML à la colonne. En réalisant cette liaison, au niveau de la définition de la colonne, toutes les informations saisies dans la colonne doivent respecter un schéma XML associé. La colonne est alors dite définie sur un type XML typé.

Les schémas XML agissent comme une sorte de contrainte d'intégrité forte, en garantissant une structure clairement identifiée à toutes les informations présentes dans cette colonne. Les mises à jour des informations XML sont également mieux contrôlées et réalisées plus rapidement dans le processus d'exécutions des requêtes.

Lors de la définition de la colonne, il est possible par l'intermédiaire des mots clés DOCUMENT et CONTENT de spécifier que la colonne ne va contenir que des documents XML bien formés, c'est-à-dire avec un seul élément au niveau supérieur. Dans le deuxième cas, CONTENT, la colonne contient des données au format XML. Si rien n'est précisé lors de la définition de la colonne, c'est le choix CONTENT qui est appliqué par défaut.

La collection de schémas XML doit être créée avant de pouvoir être référencée par une colonne XML. La gestion des collections de schémas passe par les instructions CREATE XML SCHEMA COLLECTION, ALTER XML SCHEMA COLLECTION et DROP XML SCHEMA COLLECTION. Chaque collection va pouvoir contenir un ou plusieurs schémas XML. Cette gestion des collections donne une gestion beaucoup plus souple des colonnes XML typées, car il est toujours possible d'ajouter un schéma XML à la collection pour pouvoir répondre à de nouvelles contraintes, fonctionnalités ou formats de gestion d'informations.

Cependant, lors de la définition du schéma, il n'est pas toujours possible de définir l'ensemble des possibilités. SQL Server supporte les déclarations any, anyAttribute et anytype dans la définition de schémas.

### Syntaxe

```
CREATE XML SCHEMA COLLECTION nomSchéma
AS définitionDuSchéma;
```

*Exemple*

*Définition d'un schéma pour définir un client :*

```
CREATE XML SCHEMA COLLECTION schemaClient AS
N'<?xml version="1.0" encoding="UTF-16"?>'+
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">'+
'<xs:complexType name="Client">'+
'<xs:sequence>'+
'<xs:element name="Nom"/>'+
'<xs:element name="Prenom"/>'+
'<xs:element name="Adresse"/>'+
'<xs:element name="CodePostal"/>'+
'<xs:element name="Ville"/>'+
'</xs:sequence>'+
'</xs:complexType>'+
'</xs:schema>';
```

```
Messages
Commande(s) réussie(s).
```

Il est maintenant possible de définir une table avec une colonne qui s'appuie sur ce typage XML.

*Exemple*

*Définition de la table Vendeurs :*

```
CREATE TABLE VENDEURS(
   id int identity(1,1) constraint pk_vendeurs primary key,
   nom nvarchar(50),
   prenom nvarchar(50),
   telephone nvarchar(14),
   email nvarchar(80),
   prospect xml(schemaClient)
);
```

```
Messages
Commande(s) réussie(s).
```

Cependant, la définition n'est pas toujours aussi simple et les informations obligatoires peuvent être complétées par des informations spécifiques dans des cas bien précis. Par exemple, dans le cas présenté ici, il peut être intéressant de compléter les informations générales d'un client par des informations ponctuelles dont les critères ne sont pas nécessairement définis par avance (un téléphone supplémentaire, des remarques sur la livraison, des goûts particuliers…). Il est possible par l'intermédiaire de la balise **any** d'adapter ainsi le schéma XML ce qui donne dans le cas présent :

```
CREATE XML SCHEMA COLLECTION schemaClientEtendu AS
N'<?xml version="1.0" encoding="UTF-16"?>'+
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">'+
'<xs:complexType name="Client">'+
'<xs:sequence>'+
'<xs:element name="Nom"/>'+
'<xs:element name="Prenom"/>'+
'<xs:element name="Adresse"/>'+
'<xs:element name="CodePostal"/>'+
'<xs:element name="Ville"/>'+
'<xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>'+
'</xs:sequence>'+
'</xs:complexType>'+
'</xs:schema>';
```

```
Messages
Commande(s) réussie(s).
```

Les attributs de la balise **any** sont :

`minOccurs`

Le nombre minimal d'éléments de ce type.

`maxOccurs`

Le nombre maximal d'informations optionnelles.

`namespace`

L'espace de nom utilisé pour la validation des éléments optionnels.

`processContents`

Ce paramètre permet d'indiquer à SQL Server comment valider les éléments optionnels par rapport à un schéma.

Les valeurs possibles pour l'attribut **`processContents`** sont :

- `skip` : les éléments ne sont pas validés par rapport à un schéma.

- `strict` : les éléments sont nécessairement validés par rapport à un schéma.
- `lax` : les éléments sont validés par rapport à un schéma uniquement si celui-ci existe dans la base.

### Union et liste de type

Les schémas XML permettent de définir les types pour les données XML. Ils permettent également de définir les valeurs possibles pour certains éléments. Parfois les valeurs possibles pour un même critère (par exemple la distance) peuvent être exprimées de plusieurs façons (système métrique ou système anglo/saxon). Le schéma XML qui prendra en compte la définition des distances devra inclure l'union des deux types pour permettre la saisie de toutes les valeurs possibles. Pour faire cette union la balise **union** sera utilisée.

L'exemple suivant illustre ce propos :

```
CREATE XML SCHEMA COLLECTION schemaDistant AS
N'<?xml version="1.0" encoding="UTF-16"?>'+
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">'+
'<xs:simpleType name="Unite">'+'<xs:union>'+
'<xs:simpleType>'+
'<xs:list>'+'<xs:simpleType>'+
'<xs:restriction base="xs:string">'+
'<xs:enumeration value="cm" />'+'<xs:enumeration value="m" />'+'<xs:enumeration value="km" />'+
'</xs:restriction>'+
'</xs:simpleType>'+'</xs:list>'+
'</xs:simpleType>'+
'<xs:simpleType>'+
'<xs:list>'+'<xs:simpleType>'+
'<xs:restriction base="xs:string">'+
'<xs:enumeration value="inch" />'+'<xs:enumeration value="foot" />'+'<xs:enumeration value="yard" />'+
'</xs:restriction>'+
'</xs:simpleType>'+'</xs:list>'+
'</xs:simpleType>'+
'</xs:union>'+'</xs:simpleType>'+
'</xs:schema>';
```

```
Messages
Commande(s) réussie(s).
```

## 2. Travailler avec une colonne de type XML

### a. Les opérations du DML

Que la colonne soit typée ou non, il est possible de manipuler les informations par l'intermédiaire des instructions du DML soit INSERT, UPDATE, DELETE et SELECT. Cependant, ces instructions se comportent avec les données de type XML comme avec les colonnes basées sur un type relationnel classique, c'est-à-dire qu'elles manipulent la totalité de l'information présente dans la colonne.

Il est possible de travailler avec des données de type date et heure (datetime, date et time) dans des documents XML sans difficulté. Pour cela, les données de type date et heures sont écrites en utilisant le format YYYY-MM-DDThh:mm:ss:nnnZ pour exprimer une heure GMT ou bien YYYY-MM-DDThh:mm:ss:nnn±hh:mm pour exprimer le décalage horaire par rapport à l'heure GMT.

Les éléments du format sont :

- YYYY : année sur 4 chiffres.
- MM : mois sur 2 chiffres.
- DD : numéro du jour dans le mois sur 2 chiffres.
- hh : heure au format 24h.
- mm : minutes.
- ss : secondes.
- nnn : fractions de seconde.

Ce format de date et heure respecte la norme ISO 8601. L'utilisation de cette norme permet de transférer facilement des données de type date et heure d'un environnement à un autre.

*Exemple*

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

```
INSERT INTO CATALOGUE(numero,page)
  VALUES(1,'<page datecreation="12102005">
             <article>
               <reference>SONYMP3</reference>
               <designation>Lecteur MP3</designation>
               <prixttc>66.90</prixttc>
             </article>
             <article>
               <reference>MUOVO</reference>
               <designation>Lecteur MP3, blanc</designation>
               <prixttc>45.50</prixttc>
             </article>
             <numero>1</numero>
           </page>');
```

Messages

```
(1 ligne(s) affectée(s))
```

*Ajout d'une ligne dans une colonne XML non typée, avec importation des informations depuis un fichier.*

```
INSERT INTO CATALOGUE(numero,page)
  SELECT 2, informations
    FROM (SELECT *
          FROM OPENROWSET(BULK 'c:\formatxml.xml', SINGLE_BLOB) AS informations)
    AS fichierXML(informations);
```

Messages

```
(1 ligne(s) affectée(s))
```

## b. Les méthodes spécifiques

Lors de la recherche d'informations, l'utilisateur peut souhaiter ne récupérer que les informations pertinentes à partir de la source XML. Pour faire une analogie, le problème est similaire à celui des données de type texte lorsque l'utilisateur souhaite ne connaître qu'un sous-ensemble de la chaîne de caractères stockée dans la base. Pour mener à bien ce traitement, SQL Server propose de nombreuses fonctions de manipulation des chaînes de caractères. Pour extraire des parties d'un document XML, SQL Server propose les méthodes query(), value(), exist() et nodes(). SQL Server dispose également de la méthode modify() pour modifier des données XML.

---

Les résultats sont affichés par défaut sous forme textuelle, mais à l'aide d'un clic sur la colonne de résultats dans la fenêtre de SQL Server Management Studio, il est possible d'appeler l'éditeur XML pour obtenir une meilleure visualisation du résultat.

---

### query()

Cette méthode accepte en paramètre une requête au format XQuery qui fait référence à la liste des nœuds du document. Elle retourne un élément XML non typé, qui est issu de la colonne de type XML.

*Exemple*

```
SELECT numero, page.query('page/article/reference')
  FROM CATALOGUE;
```

Résultats | Messages

| | numero | (Aucun nom de colonne) |
|---|---|---|
| 1 | 1 | <reference>SONYMP3</reference><reference>MUOVO</reference> |

## value()

Cette méthode retourne une valeur scalaire issue du document XML. Elle accepte en paramètre une requête XQuery et le nom du type de données de la valeur retournée.

*Exemple*

*L'exemple suivant permet de connaître la référence et le prix du 1er article présent dans le document XML qui porte le numéro 1.*

```
SELECT numero,
       page.value('(page/article/reference)[1]','nvarchar(10)'),
       page.value('(page/article/prixttc)[1]','money')
  FROM CATALOGUE
 WHERE numero=1;
```

Résultats | Messages

| | numero | (Aucun nom de colonne) | (Aucun nom de colonne) |
|---|---|---|---|
| 1 | 1 | SONYMP3 | 66,90 |

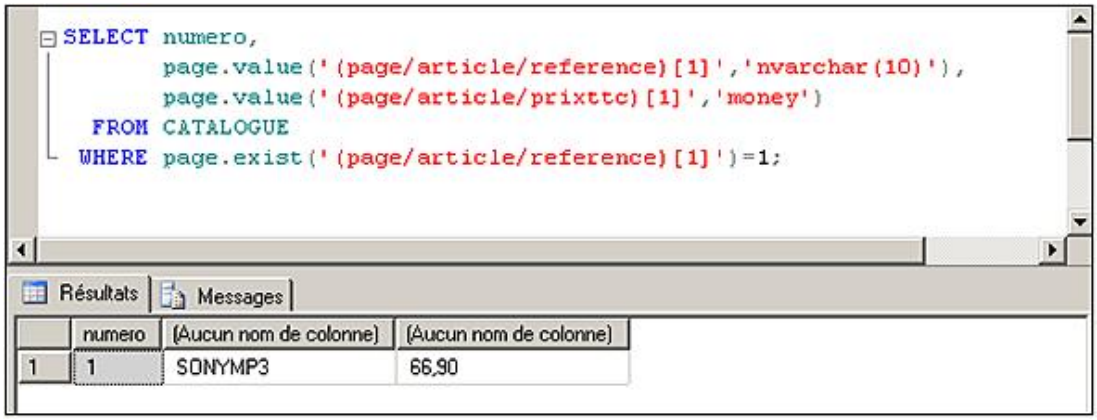
## exits()

Cette méthode qui accepte en paramètre une requête de type XQuery, retourne une valeur scalaire de type bit :

- 1 si la requête XQuery permet de localiser de l'information.
- 0 si la requête XQuery ne retourne pas d'information.
- NULL si la colonne XML sur laquelle porte la fonction contient la valeur NULL.

Il est utile de travailler avec cette méthode conjointement à la méthode value afin de ne pas remonter d'erreur lorsque la requête XQuery n'aboutit pas à un résultat.

*Exemple*




## nodes()

Cette méthode est utile dans le cas où l'objectif recherché est un mappage entre le type XML et un stockage des informations au format relationnel. En effet, cette méthode permet, à partir d'une requête XQuery passée en paramètre, d'obtenir autant de lignes que de nœuds définis à ce niveau. Chaque ligne du jeu de résultats est au format XML, mais il est ainsi facile de morceler proprement un document XML.

*Exemple*

*À partir d'une donnée de type XML stockée dans une variable, les informations sont découpées par rapport à la requête XQuery passée en paramètre à la méthode nodes. Le jeu de résultats est ensuite affiché, mais il est également possible de le stocker dans une colonne de type XML d'une table de la base de données.*


## modify()

La méthode **modify** permet de modifier une partie des données stockées dans un document xml. Contrairement à l'instruction update qui permet de mettre à jour le contenu global d'une colonne, la méthode **modify** permet de modifier des valeurs directement dans la colonne de type xml. La méthode **modify** permet l'ajout de valeurs, la modification de valeurs existantes ou bien la suppression de valeurs. Pour réaliser ces trois opérations, la méthode **modify** utilise les expressions XML DML : insert, replace value of et delete.

*Exemple*

*Dans l'exemple suivant, le document xml initial est modifié en y ajoutant une ligne d'information :*


## 3. Indexer une colonne de type XML

SQL Server offre la possibilité de définir des index sur les colonnes de type XML. Lors d'une requête, les informations XML sont traitées au niveau de chaque ligne, ce qui peut entraîner des traitements longs et coûteux surtout lorsque le nombre de lignes concernées par la requête est important et/ou quand les informations au format XML sont volumineuses.

Le mécanisme habituellement utilisé pour définir un index sur des données relationnelles repose sur la structure d'un arbre balancé, c'est-à-dire que lors de la construction d'index, le nombre de questions pour arriver au niveau feuille de l'index est toujours le même quelle que soit la valeur recherchée. Ce type de structuration est également adopté pour définir l'index dit principal sur la colonne de type XML. Cet index est, entre autres, composé des balises et des valeurs contenues dans la colonne de type XML. À partir de cet index principal, il est possible de définir des index, dits cette fois-ci secondaires, afin d'accélérer le traitement des requêtes. Ces index secondaires sont définis par rapport à des classes de requêtes fréquentes :

- l'index PATH pour des requêtes portant sur le chemin d'accès ;
- l'index PROPERTY pour des requêtes portant sur les propriétés ;
- l'index VALUE pour des requêtes portant sur des valeurs.

---

Il est également possible de définir un index de texte intégral sur les colonnes de type XML, pour indexer uniquement le contenu en ignorant les balises XML.

---

### a. Index principal

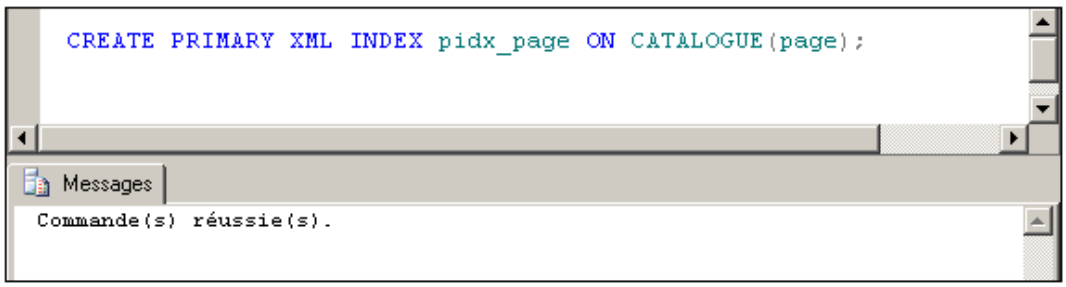
L'index principal sur une colonne de type XML nécessite que la table possède un index organisé sur la clé primaire.

*Syntaxe*

```
CREATE PRIMARY XML INDEX nomIndex ON table(colonneXML)
```

*Exemple*

*Un index principal est défini sur la colonne page de la table catalogue.*



### b. Index secondaire

Il n'est possible de créer un index secondaire que si un index principal est défini pour la colonne.

Le document XML ne peut contenir que 128 niveaux au maximum. Les documents qui possèdent une hiérarchie plus complexe sont rejetés lors de l'insertion ou de la modification des colonnes.

De même, l'indexation porte sur les 128 premiers octets du nœud, les valeurs plus longues ne sont pas prises en compte dans l'index.

*Syntaxe*

```
CREATE XML INDEX nomIndex ON table(colonneXML)
USING XML INDEX nomIndexPrincipal FOR {PATH|PROPERTY|VALUE}
```

PATH

Permet de construire un index sur les colonnes path et value (chemin et valeur) de l'index XML primaire. Un tel index pourra améliorer sensiblement les réponses lors de l'utilisation de la méthode **exist()**, dans une clause where par exemple.

PROPERTY

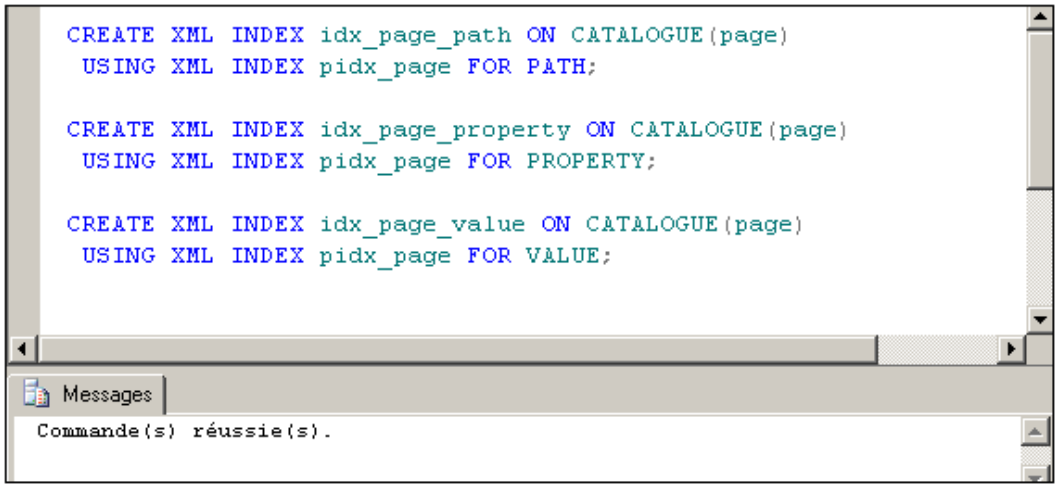
Permet de construire un index sur les colonnes PK, path et value de l'index XML principal. Le symbole PK correspondant à la clé primaire de la table. Ce type d'index est donc utile lors de l'utilisation de la méthode **value()** dans les requêtes SQL de manipulation des données.

VALUE

Permet de construire un index sur les colonnes value et path de l'index XML principal. Cet index profite principalement aux requêtes pour lesquelles la valeur du nœud est connue indépendamment du chemin d'accès, ce qui peut être le cas lors de l'utilisation de la méthode **exist()** par exemple.

*Exemple*

*Dans l'exemple ci-après, les trois types d'index sont créés par rapport à l'index XML principal défini sur la colonne page de type XML de la table Catalogue.*




## 4. XQuery et Xpath

Les documents au format XML sont de plus en plus nombreux. Il est donc de plus en plus fréquent d'être conduit à rechercher de l'information dans ces sources de données. Le langage XQuery a pour objectif de répondre à cette problématique en fournissant un moyen simple et performant pour interroger les données au format XML. L'objectif de XQuery est de fournir un langage aussi simple et performant pour mener des interrogations sur des données au format XML que le langage SQL par rapport aux données stockées au format relationnel.

Il est possible de consulter la documentation exacte des règles qui régissent le langage d'interrogation XQuery en

visitant le site web suivant : http://www.w3.org/TR/xquery.

Le Transact SQL de SQL Server implémente seulement un sous ensemble de XQuery tel qu'il est défini par le W3C. Ce sous-ensemble permet de réaliser sans aucun soucis la plupart des requêtes. L'implémentation XQuery dans SQL Server donne, entre autres, la possibilité de construire des boucles for, de faire des restrictions (where), de retourner une valeur (return) et de trier le jeu de résultats (order by).

SQL Server propose également la fonction let qui permet de valoriser des variables dans le contexte d'une requête XQuery.

Le langage de navigation XPath 2.0 est intégré dans XQuery. Il correspond à des requêtes XQuery qui sont définies uniquement et entièrement avec un chemin. L'expression de ce chemin permet de localiser un ou plusieurs nœuds dans le document XML d'origine. Il est possible de limiter le nombre de nœuds sélectionnés en définissant des filtres. Ces filtres peuvent utiliser les valeurs contenues dans les attributs des nœuds pour évaluer une condition booléenne.

Les expressions XQuery sont toujours structurées de la même façon : un en-tête (prologue) et une expression (body).

L'en-tête est souvent omis pour les requêtes définies sur SQL Server.

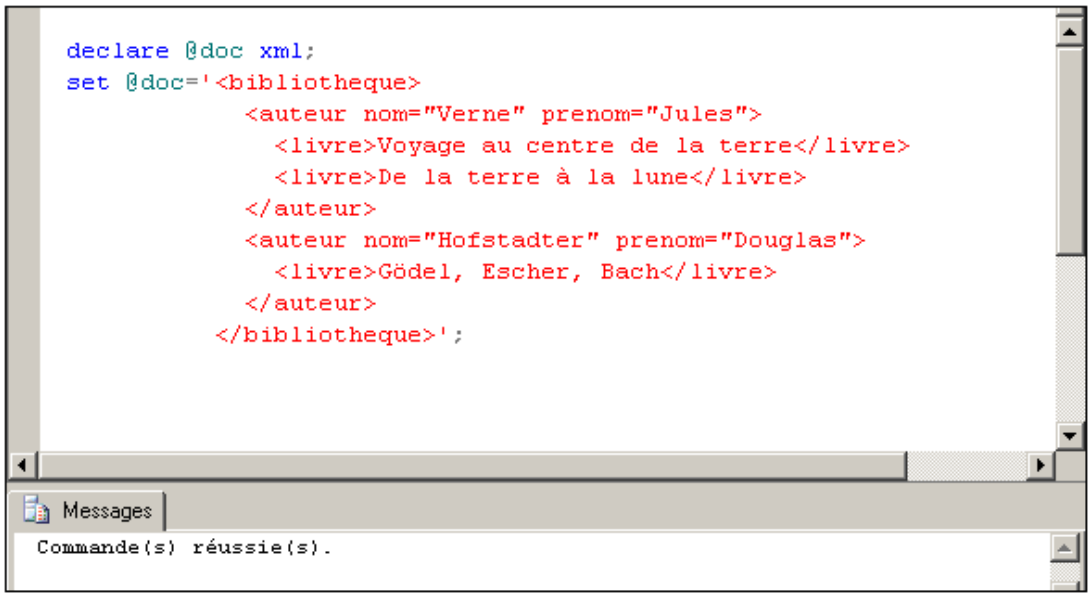
L'expression XQuery représente la requête proprement dite. Le langage XQuery est fortement typé et sensible à la casse, ce qui est cohérent avec XML mais inhabituel par rapport au SQL. Les mots clés sont toujours en minuscule.

La requête XQuery la plus simple correspond simplement à un chemin relatif au document XML. Ce type de requêtes peut, par exemple, être utilisé en paramètre de la méthode **query()**.

Le chemin est exprimé par rapport à la racine de la structure XML. Le résultat de la requête correspond à tous les éléments qui possèdent le niveau de hiérarchie exprimé dans la requête.

*Exemple*

*Soit la variable de type XML suivante :*




*La requête XQuery bibliotheque/auteur fournira comme résultat l'ensemble des auteurs et de leurs livres.*

*La requête XQuery bibliotheque/auteur/* fournira quant à elle uniquement les livres car le chemin des éléments sélectionnés doit être bibliotheque/auteur/quelquechose.*

*Enfin, la requête bibliotheque/auteur/livre/* ne fournira aucun résultat car le nœud livre n'est composé d'aucun sous-nœud.*

```
----------> Requete 1
select @doc.query('bibliotheque/auteur');
----------> Requete 2
select @doc.query('bibliotheque/auteur/*');
----------> Requete 3
select @doc.query('bibliotheque/auteur/livre/*');
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | <auteur nom="Verne" prenom="Jules"><livre>Voyage... |

| | (Aucun nom de colonne) |
|---|---|
| 1 | <livre>Voyage au centre de la terre</livre><livr... |

| | (Aucun nom de colonne) |
|---|---|
| 1 | |

La requête XQuery peut également permettre de localiser une valeur en particulier. Dans ce cas, l'expression du chemin de l'élément va être accompagnée de filtres afin de définir des conditions supplémentaires sur la sélection des lignes. Ce filtre va correspondre au numéro de l'instance du nœud qui doit être sélectionné. Ce numéro est exprimé entre crochets [ ].

*Exemple*

*La requête XQuery bibliotheque/auteur[1]/* fournira la liste des livres dont l'auteur est le premier à être référencé dans le document XML, soit Jules Verne dans l'exemple présenté ici.*

*La requête XQuery bibliotheque/auteur[2]/* fournira la liste des livres dont l'auteur est le second à être référencé dans le document XML, soit Douglas Hofstadter.*

```
----------> Requete 1
select @doc.query('bibliotheque/auteur[1]/*');
----------> Requete 2
select @doc.query('bibliotheque/auteur[2]/*');
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | <livre>Voyage au centre de la terre</livre><livr... |

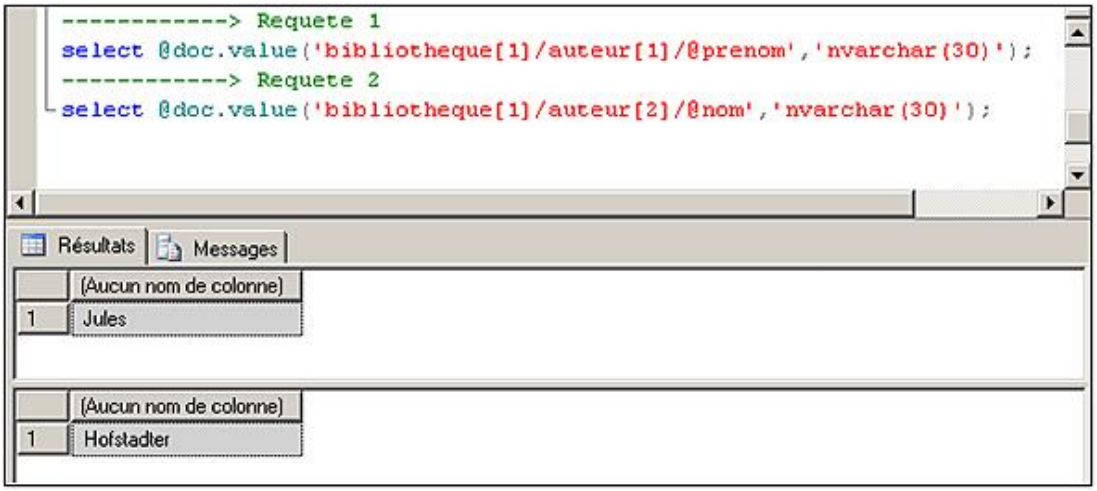
| | (Aucun nom de colonne) |
|---|---|
| 1 | <livre>Gödel, Escher, Bach</livre> |

Il est possible de travailler avec les attributs des nœuds XML dans la requête XQuery. Pour référencer un attribut, il faut faire précéder son nom du symbole @.

*Exemple*

*Pour afficher le prénom des auteurs, il faudra utiliser la requête XQuery suivante : bibliotheque/auteur/@prenom. Afin d'être certain que la requête va ramener une valeur unique, il faut préciser le numéro de chaque nœud présent dans la requête.*

*L'exemple permet de connaître le prénom du premier auteur et le nom du second.*




Il est possible de travailler avec les paramètres pour poser des conditions de sélection des nœuds. Le filtre exprimé entre les crochets permet de définir une condition qui va retourner une valeur booléenne afin de savoir si le nœud est sélectionné ou non.

*Exemple*

*La requête XQuery bibliotheque/auteur[@prenom='Jules']/* va permettre de connaître la liste des livres écrits par un auteur dont le prénom est Jules.*

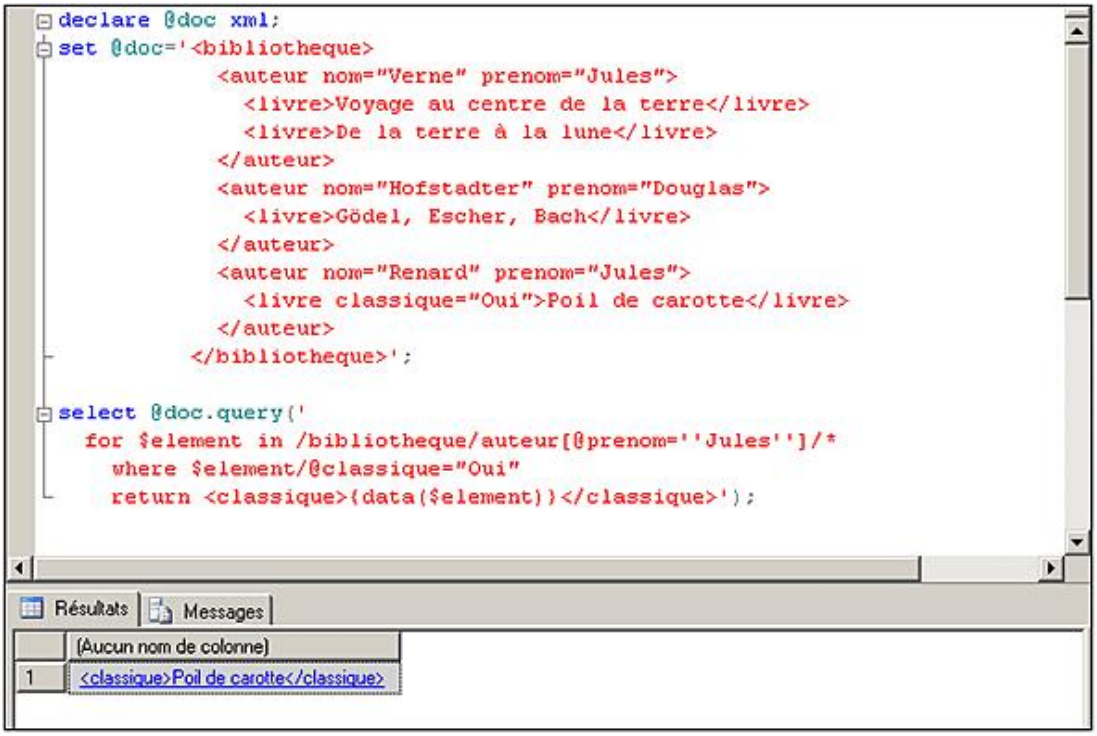

Pour écrire des requêtes d'extraction de données encore plus souples, XQuery propose les instructions for, where et return. La construction de la requête peut alors devenir complexe et permettre d'extraire les informations de façon propre. La requête XQuery contient tous les éléments pour extraire l'information du document au format XML.

*Exemple*

*Dans l'exemple suivant, le document XML est parcouru au niveau des auteurs qui possèdent le prénom Jules. Ce parcours est assuré par la boucle for. Chaque sous-ensemble du document qui correspond à un pas du parcours est conservé dans la variable $element.*

*Pour chaque élément de la boucle for, une condition supplémentaire est posée par l'opérateur where. Dans l'exemple présenté ici, seuls les livres identifiés comme classiques sont conservés.*

*Enfin, les informations relatives aux livres sont retournées comme un élément du nœud classique, par l'intermédiaire de l'instruction return :*



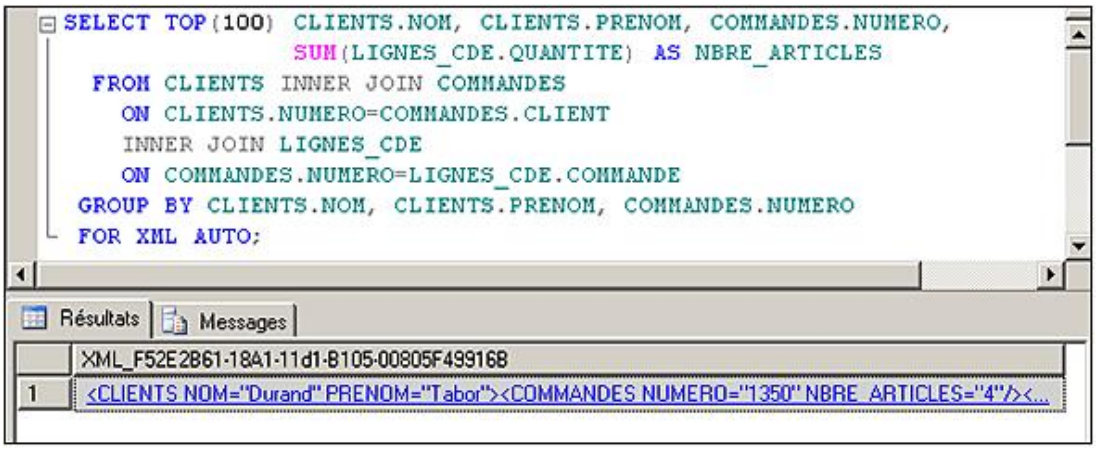

Lors du parcours du document xml, il est parfois nécessaire d'affecter des valeurs à des variables. Cette opération est possible par l'intermédiaire de l'instruction let qui permet la valorisation d'une variable dans une expression XQuery. La requête XQuery est alors en mesure de réaliser des opérations plus complexes comme le dénombrement.

## 5. FOR XML

L'utilisation de FOR XML représente un bon moyen de gérer l'échange au format XML vers une application cliente. Cette clause permet de convertir le résultat d'une requête SELECT au format XML.

*Exemple*

*Cette requête établit la liste des commandes par clients au format XML :*




*Le résultat au format XML de cette requête est alors :*

```
<CLIENTS NOM="Durand" PRENOM="Tabor">
  <COMMANDES NUMERO="1350" NBRE_ARTICLES="4" />
</CLIENTS>
<CLIENTS NOM="Gregoire" PRENOM="Maryse">
  <COMMANDES NUMERO="1351" NBRE_ARTICLES="40" />
</CLIENTS>
<CLIENTS NOM="Blondlot" PRENOM="Lowell">
  <COMMANDES NUMERO="1352" NBRE_ARTICLES="24" />
</CLIENTS>
<CLIENTS NOM="Boivin" PRENOM="Céline">
  <COMMANDES NUMERO="1353" NBRE_ARTICLES="14" />
</CLIENTS>
<CLIENTS NOM="Charpie" PRENOM="Natalie">
  <COMMANDES NUMERO="1354" NBRE_ARTICLES="16" />
</CLIENTS>
<CLIENTS NOM="Chassé" PRENOM="Astolpho">
```

Avec l'apparition du type XML et la demande de plus en plus présente de fournir des données au format XML, la clause FOR XML a été remaniée avec SQL Server 2005. L'objectif recherché est de pouvoir extraire l'information depuis la base relationnelle vers un format XML sans recourir à des outils tiers de manipulation et de transformation des données.

Tout d'abord, et par souci de compatibilité ascendante, les résultats de la clause FOR XML sont toujours fournis au format texte par défaut. Cependant, l'option TYPE permet d'obtenir le même résultat mais exprimé au format XML cette fois-ci. Cette option est particulièrement intéressante lorsque le résultat de la requête doit être stocké dans une colonne de type XML par exemple.

*Exemple*

*La requête précédente est modifiée pour obtenir un résultat au format XML :*

```
SELECT TOP(100) CLIENTS.NOM, CLIENTS.PRENOM, COMMANDES.NUMERO,
                SUM(LIGNES_CDE.QUANTITE) AS NBRE_ARTICLES
  FROM CLIENTS INNER JOIN COMMANDES
    ON CLIENTS.NUMERO=COMMANDES.CLIENT
    INNER JOIN LIGNES_CDE
    ON COMMANDES.NUMERO=LIGNES_CDE.COMMANDE
 GROUP BY CLIENTS.NOM, CLIENTS.PRENOM, COMMANDES.NUMERO
 FOR XML AUTO, TYPE;
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | <CLIENTS NOM="Durand" PRENOM="Tabor"><COMMANDES N... |

Cette directive TYPE permet également de simplifier l'écriture des requêtes SQL qui ont en charge de générer un document XML plus ou moins complexe. Car il est possible d'utiliser toutes les fonctionnalités du SQL pour générer rapidement et simplement un document au format XML.

*Exemple*

*Imbrication de requêtes qui retournent une valeur au format XML :*

```
SELECT TOP(100) NUMERO AS CLIENT,
                (SELECT NUMERO AS COMMANDE
                   FROM COMMANDES
                  WHERE COMMANDES.CLIENT=CLIENTS.NUMERO
                  FOR XML AUTO, TYPE)
FROM CLIENTS
FOR XML AUTO, TYPE;
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | <CLIENTS CLIENT="25001" /><CLIENTS CLIENT="16000... |

*Le résultat visible au travers de l'éditeur XML est :*

```
<CLIENTS CLIENT="25001" />
<CLIENTS CLIENT="160001">
  <COMMANDES COMMANDE="1645" />
</CLIENTS>
<CLIENTS CLIENT="160003" />
<CLIENTS CLIENT="160004" />
<CLIENTS CLIENT="160005" />
<CLIENTS CLIENT="160006" />
<CLIENTS CLIENT="160007" />
<CLIENTS CLIENT="160008" />
<CLIENTS CLIENT="160009" />
<CLIENTS CLIENT="160010" />
<CLIENTS CLIENT="160011" />
<CLIENTS CLIENT="160012" />
<CLIENTS CLIENT="160013" />
<CLIENTS CLIENT="160014" />
```

Il existe également la possibilité de définir un chemin (PATH) au niveau de la clause FOR XML, ou bien directement au niveau de l'alias de colonne dans la requête Select.

*Exemple*

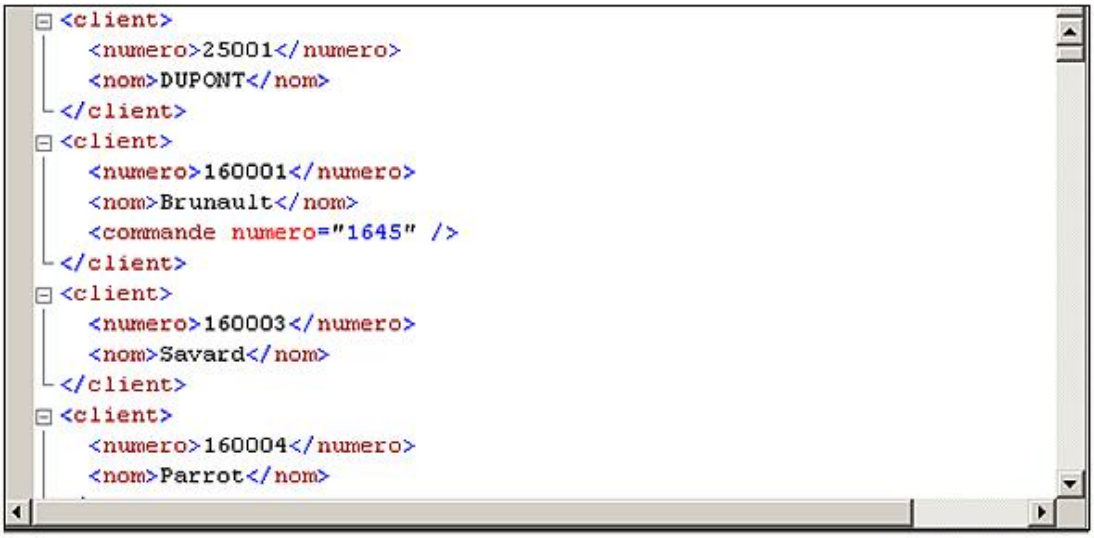
*Le cas précédent a été remanié en précisant les chemins afin d'obtenir exactement le document XML souhaité :*

```
SELECT TOP(100) NUMERO AS "numero",
                NOM as "nom",
                (SELECT NUMERO AS "@numero"
                   FROM COMMANDES
                  WHERE COMMANDES.CLIENT=CLIENTS.NUMERO
                  FOR XML PATH('commande'), TYPE)
FROM CLIENTS
FOR XML PATH('client'), TYPE;
```

Résultats | Messages

| | (Aucun nom de colonne) |
|---|---|
| 1 | <client><numero>25001</numero><nom>DUPONT</nom><... |

*Le résultat au format XML est alors :*




## 6. OpenXML

Cette méthode permet de traiter un document XML sous la forme d'un jeu de résultats. Avec OPEN XML, il est possible d'intégrer le traitement d'un document XML dans une requête SQL de type SELECT, INSERT, UPDATE et DELETE.

---

Avec l'apparition du type XML, cette méthode tend à être de moins en moins utilisée.

---

**Syntaxe**

```
OPENXML(idoc int ,requeteXPath nvarchar,[drapeau byte])
[WITH (structureDonnées)]
```

`idoc`

Identifiant du document XML. Cet identifiant est fourni par la procédure sp_xml_preparedocument.

`requeteXPath`

Requête au format XPath pour localiser les informations dans le document XML.

`drapeau`

Indicateurs de paramétrages OPENXML.

`structureDonnées`

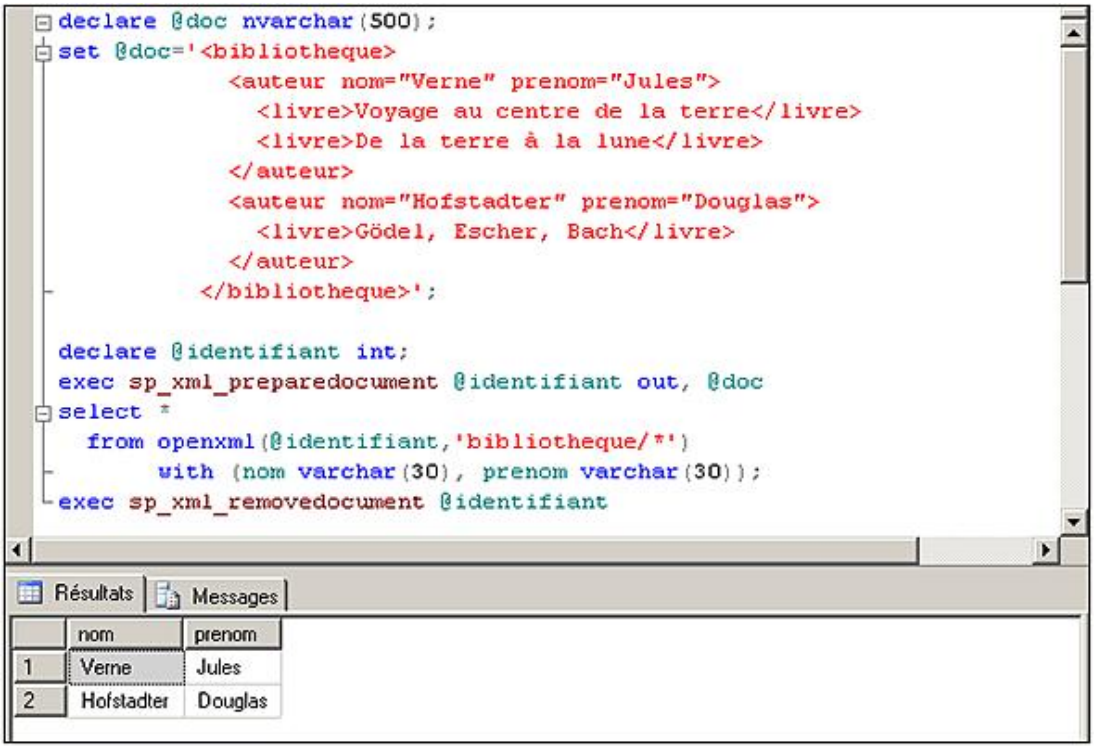
Nom et types des colonnes à présenter dans le jeu de résultats.

La méthode sp_xml_preparedocument permet de préparer un document au format texte vers le format XML. L'identifiant du résultat de cette transformation est retourné par la procédure stockée. L'identifiant servira à OPENXML pour localiser les données au format XML.

En fin d'utilisation, il faut veiller à exécuter sp_xml_removedocument pour libérer l'espace mémoire occupé par le document au format XML.

*Exemple*

*Mise en place de la méthode OPENXML.*




## 7. OPENROWSET

Cette méthode permet d'accéder facilement à des ressources externes au serveur pour travailler avec ces données distantes comme si elles étaient présentes sous forme de table dans la base de travail courante. La source de données peut être le système de fichier Windows, ou une source OLEDB.

L'utilisation de cette méthode peut parfois remplacer avantageusement l'utilisation du programme d'importation bcp.

La méthode OPENROWSET présente une alternative à OPENXML pour intégrer facilement des données XML externes au serveur.

**Syntaxe**

```
OPENROWSET(nomFournisseur,chaineConnexion,requete)
```

`nomFournisseur`

Nom du fournisseur OLEDB à utiliser.

`chaineConnexion`

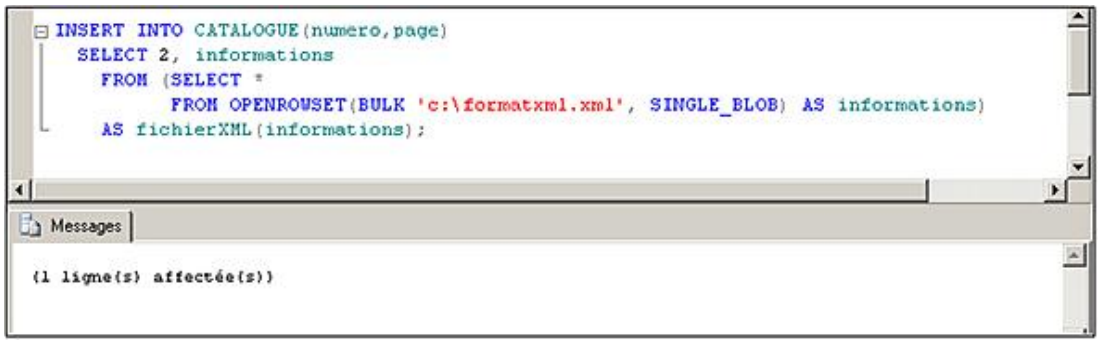
Chaîne de connexions pour pouvoir se connecter à la source de données. Les paramètres de cette chaîne sont fonction du fournisseur OLEDB sélectionné.

`requete`

Requête SQL à exécuter pour extraire l'information.

*Exemple*

*Chargement de données stockées dans un fichier plat dans le système de fichiers Windows :*



# Table value Parameter

Il s'agit d'un nouveau type de paramètre introduit avec SQL Server 2008. Avec ce type de paramètre il est possible d'envoyer un ensemble de données directement à une procédure ou bien à une fonction Transact SQL. Pour mieux comprendre l'intérêt des "table value parameter", il est possible de rapprocher ce type de paramètre à la notion de tableau. Chaque ligne de ce tableau est définie par rapport à un type de données utilisateur. Ces types sont créés par l'intermédiaire de l'instruction CREATE TYPE qui permet maintenant de définir des types TABLE. Chaque champ est fortement typé grâce à l'usage, entre autres, des contraintes d'intégrité lors de la définition du type.

Avec les table value parameter il est possible de gérer un ensemble structuré de données sans qu'il soit nécessaire de créer une table, même temporaire. En ce sens, l'utilisation de ces types permet de gagner en souplesse d'utilisation et parfois même en performance. Cependant, les table value parameter sont toujours des paramètres en lecture seule. Donc, la procédure ou la fonction qui possède un paramètre de ce type ne peut modifier les informations présentes dans ce paramètre.

L'utilisation d'un table value parameter peut se décomposer en trois étapes :

- définir le type de chaque ligne à l'aide de l'instruction CREATE TYPE ;
- compléter le table value parameter avec des données en utilisant les instructions du DML : INSERT, UPDATE et DELETE ;
- appeler la fonction ou la procédure en lui passant en paramètre le table value parameter.

Dans la procédure ou la fonction qui possède un paramètre de ce type, les données sont extraites par l'intermédiaire de requêtes SELECT ou bien par l'utilisation d'un curseur.

*Exemple*

*Dans l'exemple suivant le type tclient est défini de façon à recevoir un numéro de client et son nom.*

*La procédure affiche permet d'afficher les données contenues dans un paramètre de type tclient.*

*Enfin, un script Transact SQL permet de stocker les numéros et les noms des clients qui habitent Paris dans une variable de type tclient.*

```
create type tclient as table(
numero int,
nom nvarchar(30));
go
create procedure affiche (@lesClients tclient readonly) as
  select * from @lesClients;
go
declare @lesParisiens tclient;
insert into @lesParisiens
select numero, nom
from clients
where ville='Paris';
exec affiche @lesParisiens;
```

Résultats | Messages

| | numero | nom |
|---|---|---|
| 1 | 160006 | Poissonnier |
| 2 | 160050 | Masson |
| 3 | 160054 | Chauvin |
| 4 | 160059 | Bourdette |
| 5 | 160061 | Hervé |
| 6 | 160070 | Moïse |
| 7 | 160090 | Bordeaux |

# Les structures hiérarchiques

La notion d'organisation hiérarchique se rencontre dans de nombreux domaines dans la vie de tous les jours et la modélisation n'en est pas toujours aisée. C'est par exemple le cas pour un organigramme d'entreprise. SQL Server 2008 propose un type de données (hierarchyId) et des méthodes afin de stocker de façon structurée cette hiérarchie. Il est également possible d'optimiser le parcours de cette hiérarchie par l'intermédiaire d'index qui permettent de parcourir rapidement l'arborescence. De plus, SQL Server offre au travers du Transact SQL des méthodes spécifiques à ce parcours d'arborescence afin de faciliter les extractions de données.

## 1. HierarchyId

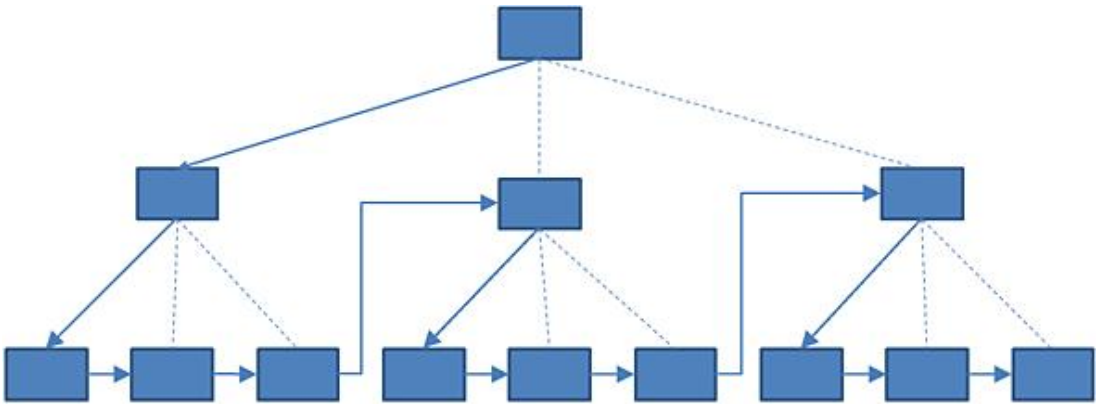
Il s'agit d'un type de données spécifique à SQL Server qui peut être utilisé pour modéliser une structure hiérarchique dans une table relationnelle. Les données pourront être extraites de cette table en utilisant les requêtes hiérarchiques.

Cette notion de hiérarchie n'est en aucun cas assimilable à une contrainte d'intégrité ; il est possible de trouver des éléments orphelins c'est-à-dire qui ne sont pas rattachés à l'arborescence définie. Les éléments orphelins peuvent apparaître suite à la saisie d'une mauvaise valeur ou suite à la suppression de l'élément qui était le supérieur hiérarchique.

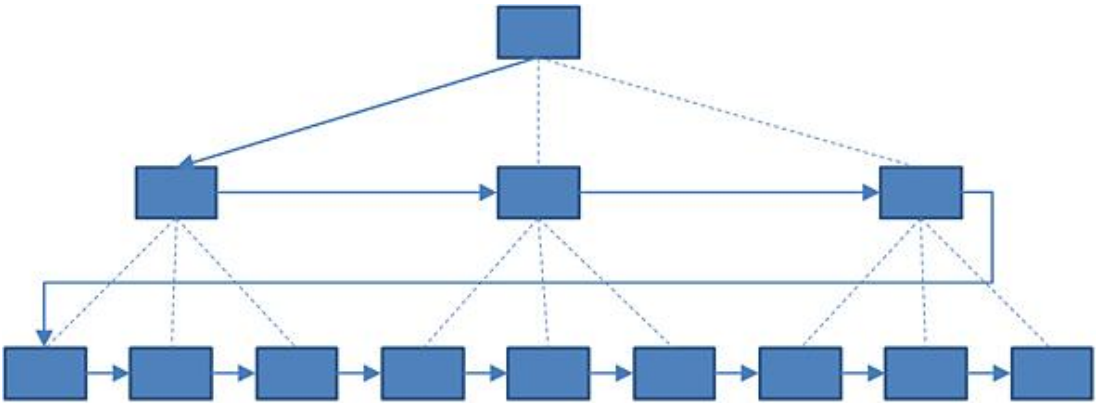
Le type hierarchyId offre tout le support nécessaire pour modéliser proprement une hiérarchie dans les tables. Toutefois, le simple fait de définir une colonne de ce type dans une table ne garantit en aucun cas que les données soient classées de façon hiérarchique. C'est au développeur ou bien à l'utilisateur final de valoriser cette colonne. De même, si la notion d'unicité des valeurs doit être gérée, il est nécessaire de définir une contrainte d'unicité.

## 2. Les index

Afin d'accélérer le parcours des arbres hiérarchiques, il est possible de définir des index. Par contre les index ne seront utiles que s'ils respectent cette arborescence. Comme il existe deux façons de parcourir un arbre, il existe deux façons de définir ces index. La première méthode est d'essayer d'aboutir le plus rapidement au niveau feuille, puis par la suite explorer l'ensemble des feuilles d'une même branche avant de passer à la branche suivante.



La seconde méthode consiste à explorer un niveau de façon complète avant de passer au parcours des éléments du niveau inférieur. Bien entendu, il est possible de définir les deux index sur la même table.



## 3. Les méthodes

`GetAncestor(niveau)`

Cette méthode permet d'identifier le supérieur hiérarchique d'un nœud. La valeur entière passée en paramètre permet de préciser le nombre de niveaux à remonter. Cette valeur est donc strictement positive et inférieure à la valeur retournée par GetLevel.

`GetDescendant([nœudFrère1[, noeudFrère2]])`

Cette méthode permet de connaître le nœud qui est le descendant direct du nœud identifié comme étant le père et par rapport auquel la méthode est exécutée. Dans le cas où il y a plusieurs nœud fils il est possible de préciser 1 ou 2 frères du descendant afin de localiser précisément l'emplacement du nœud descendant. Ce type de positionnement est particulièrement utile lors de l'insertion de données dans un arbre trié.

`GetLevel()`

Cette méthode est nécessaire lors de la définition d'un index BreadthFirst. Elle permet de connaître la profondeur d'un nœud dans l'arborescence.

`GetRoot()`

Cette méthode permet d'identifier de façon immédiate le nœud racine de l'arborescence.

`IsDescendant(noeudATester)`

Cette méthode permet de déterminer rapidement si le nœud passé en paramètre appartient à la descendance du nœud par rapport auquel cette méthode est exécutée.

`Parse(chaineCaractères)`

Cette méthode permet de convertir la chaîne de caractères passée en paramètre en une valeur de type hierarchyId.

`Read(lecteurBinaire)`

Cette méthode n'est utilisable que dans le cadre du CLR et permet de faire une lecture en binaire du nœud par rapport auquel cette méthode est exécutée.

`Reparent(ancienneRacine, nouvelleRacine)`

Cette méthode permet de modifier le nœud racine et de définir le nouveau chemin d'accès par rapport à la nouvelle racine passée en paramètre.

`ToString()`

Cette méthode est la méthode inverse de Parse et permet d'obtenir une représentation textuelle de la valeur hierarchyId du nœud.

`Write(fluxBinaire)`

Cette méthode réservée au code CLR permet de définir à partir d'un flux binaire la valeur de type hierarchyId du nœud. La méthode Write permet de réaliser l'opération inverse du travail réalisé avec la méthode Read.

*Exemple*

*La première étape consiste à créer une table avec une colonne de type hierarchyId. Dans l'exemple présenté ci-dessous, une table représentant les employés d'une entreprise avec leur poste et leur position hiérarchique respective est définie :*

```
create table employes(
  id int identity (1,1),
  position hierarchyId,
  nom nvarchar(80),
  prenom nvarchar(80),
  poste nvarchar(80));
```

Messages

```
Commande(s) réussie(s).
```

*Les index sont ensuite définis sur cette table. Le premier index est créé par l'intermédiaire de l'ajout d'une contrainte de clé primaire. Le second index concerne la colonne de type hierarchyId. Enfin, pour permettre de parcourir rapidement tous les employés se trouvant à un même niveau dans la hiérarchie, une nouvelle colonne est ajoutée à la définition de la table. Cette colonne contient une donnée calculée à partir de la position de l'employé dans la hiérarchie :*

```
alter table employes
 add constraint pk_employes primary key (id);
create index employes_position
  on employes(position);
-- modifier la table pour ajouter une colonne relative au niveau
alter table employes
  add niveau as position.GetLevel();
create index employes_niveau
  on employes(niveau, position);
```

Messages

```
Commande(s) réussie(s).
```

*Maintenant que la structure est définie, il est possible d'ajouter des informations dans cette table :*

```
-- Ajouter le sommet de la hiérarchie
insert into employes (position, nom,prenom, poste)
  values (hierarchyId::GetRoot(), 'DUPONT','Emile','Directeur');

declare @patron hierarchyid;
select @patron=hierarchyid::GetRoot() from employes;
-- Ajouter le second niveau
declare @drh hierarchyid;
declare @compta hierarchyid
set @drh=@patron.GetDescendant(null, null);
insert into employes (position, nom, prenom, poste)
  values  (@drh,'BARTIN','Jeanne','DRH');
set @compta=@patron.GetDescendant(@drh, null);
insert into employes (position, nom, prenom, poste)  values
  ( @compta,'MICHALON','Paul','Comptable');
-- ajouter un troisième niveau
insert into employes (position, nom, prenom, poste) values
  (@drh.GetDescendant(null, null),'BERNAUD','Beatrice','Assistante');
insert into employes ( position, nom, prenom, poste) values
  (@compta.GetDescendant(null, null),'MARBOT','Marcel','Assistant');
```

Messages

```
(1 ligne(s) affectée(s))
```

*Il est maintenant possible d'extraire les informations comme on le souhaite. Dans ce premier exemple, tous les employés*

*qui ont pour supérieur hiérarchique direct le patron sont extraits :*

```
declare @chef hierarchyid;
select @chef=hierarchyid::GetRoot() from employes

select *
  from employes
  where position.GetAncestor(1)=@chef;
```

Résultats | Messages

| | id | position | nom | prenom | poste | niveau |
|---|---|---|---|---|---|---|
| 1 | 2 | 0x58 | BARTIN | Jeanne | DRH | 1 |
| 2 | 3 | 0x68 | MICHALON | Paul | Comptable | 1 |

# Les données non structurées

Aujourd'hui, les bases de données doivent être en mesure de stocker des données non structurées et c'est ce que propose SQL Server avec son type filestream. En effet, les documents numériques sont de plus en plus présents dans notre quotidien et aujourd'hui il est courant de travailler avec des photos, des fichiers word, excel, des documents scannés… Or tous ces documents ne peuvent que difficilement trouver leur place dans une base de données relationnelle structurée avec des types de données simples. De plus, ces documents représentent très souvent un volume important. C'est pourquoi, bien souvent l'une des options suivantes est retenue :

- les données structurées sont stockées dans la base de données tandis que les données non structurées (fichiers) sont stockées directement sur le système de fichiers,
- les données structurées sont stockées dans une base de données et les données non structurées sont stockées dans une autre base,
- toutes les données, structurées ou non, sont stockées dans la base de données.

Les deux premières solutions posent le problème de la liaison entre les différentes données. Comment, par exemple, associer correctement l'image d'un produit à sa référence, sa désignation… et plus particulièrement comment garantir que lors de la suppression d'une image, l'article associé n'existe plus.

La troisième solution évite ces problèmes mais pose le problème délicat de la gestion de l'espace disque dans la base de données. De plus, les fichiers de données très volumineux ont tendance à dégrader les performances du moteur de base de données.

Pour essayer de tirer parti des différentes options, SQL Server propose deux solutions différentes.

La première consiste à utiliser une seconde base pour stocker les données de type blob (*Binary Large OBject*) mais tout en conservant une liaison entre les données locales et les données distantes. Ce serveur de fichiers BLOB peut bien sûr être d'origine Microsoft mais tous les grands acteurs du marché sont compatibles avec cette solution. En effet, ce type de solution peut être intéressant lorsque les données de type BLOB doivent être stockées sur un serveur distinct du serveur SQL, ou bien lorsqu'il existe déjà des applications qui alimentent ce magasin de BLOB.

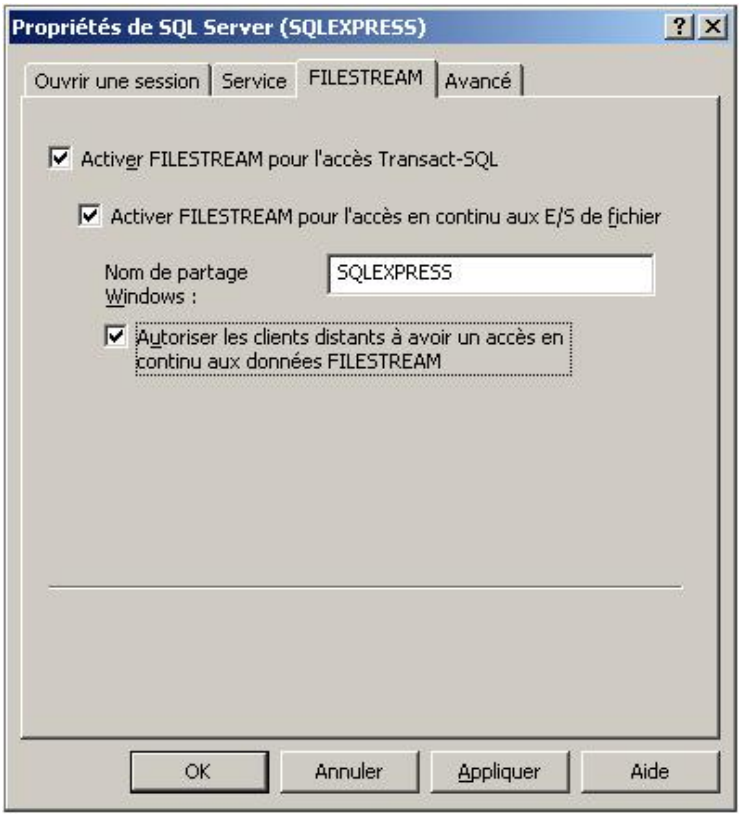
La seconde possibilité est de définir une colonne de type varbinary en utilisant l'attribut FILESTREAM qui permet de stocker le fichier sur le système de fichiers tout en le liant avec les données relationnelles. Ce mode de stockage permet d'utiliser les avantages de chaque solution sans en avoir les inconvénients. Les données stockées dans une colonne FILESTREAM sont perçues par SQL Server comme des données de type BLOB et peuvent être manipulées au sein des requêtes comme n'importe quelle donnée relationnelle stockée dans la base. De plus, cette colonne reçoit un traitement identique aux autres colonnes BLOB lors des opérations de maintenance, de sauvegarde, de restauration.

Ce type de colonne impose tout même quelques limitations, comme le stockage des informations sur un disque local, le cryptage automatique des données n'est pas supporté de même que la mise en miroir. Il n'est pas possible également de définir une colonne de type FILESTREAM dans un paramètre table-value.

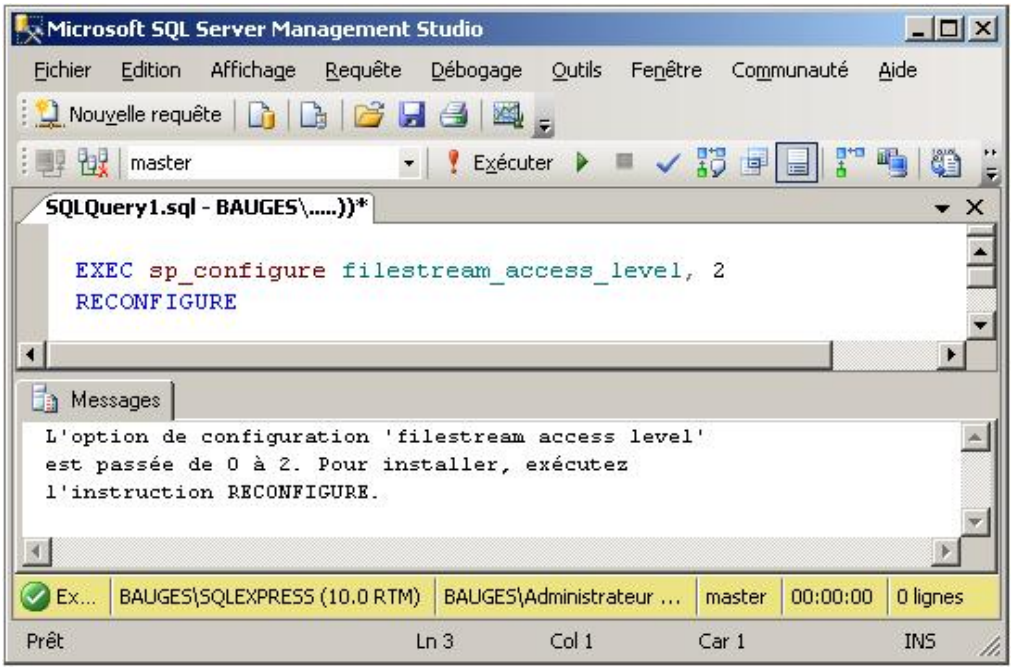
Quel que soit le mode de stockage retenu, la recherche dans ces données reste toujours difficile. Pour réaliser ce travail, SQL Server utilise le service de texte intégral qui est intégré à SQL Server. Cette intégration permet de réduire la complexité de gestion et donc le coût lors du déploiement de solutions SQL Server utilisant cette fonctionnalité. De plus, cette intégration permet d'obtenir de meilleures performances.

La mise en place de la prise en charge du type FILESTREAM se déroule en deux temps qui sont la configuration de ce type au niveau de l'instance puis l'activation.

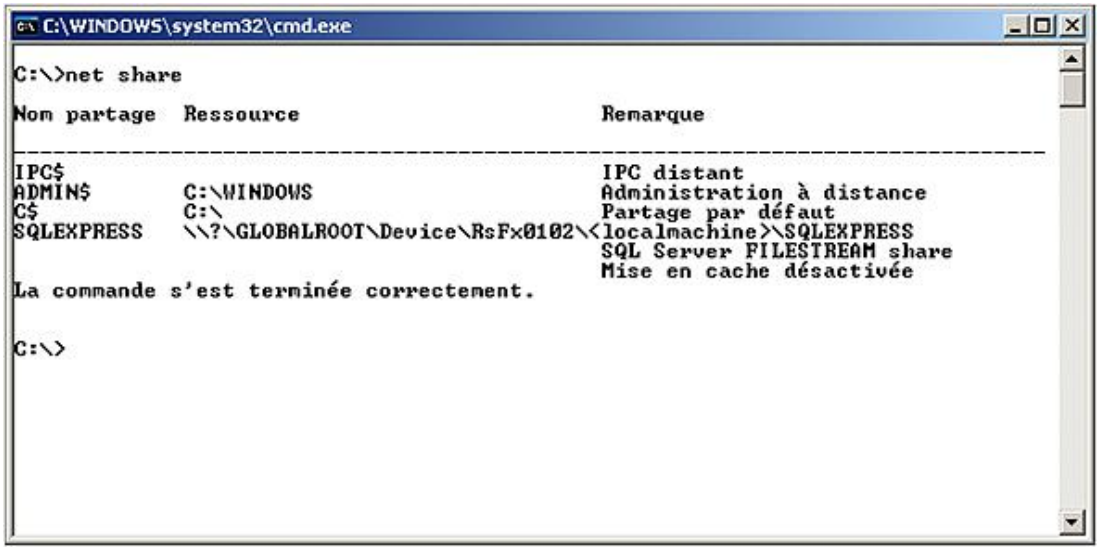
La configuration au niveau de l'instance s'effectue à partir de la boîte de dialogue des **Propriétés** de l'instance depuis le **Gestionnaire de configuration de SQL Server**. Cette boîte de dialogue, présentée ci-dessous, permet d'activer la prise en charge du type FILESTREAM.




Il ne reste plus alors qu'à configurer l'instance pour la prise en charge de ce nouveau type :



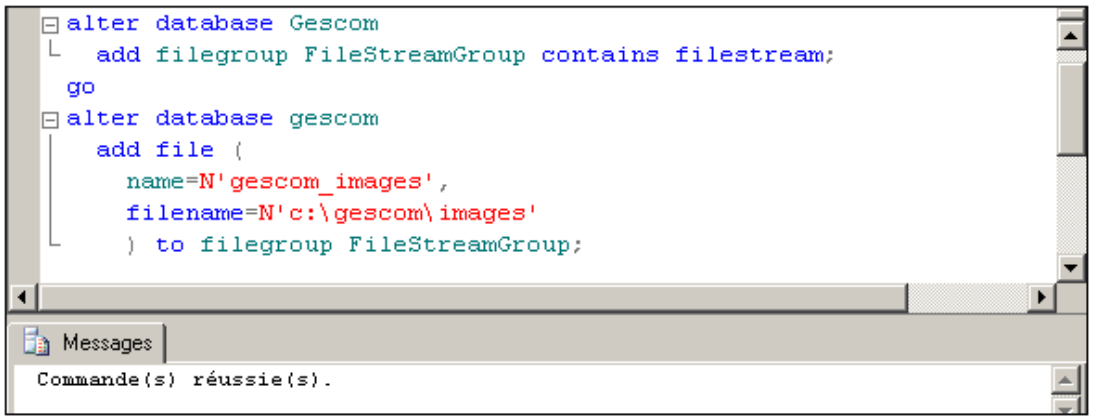

*Il est possible de vérifier le résultat de cette commande en exécutant l'instruction net share depuis l'interpréteur de commande Windows :*




Pour gérer les fichiers physiques SQL Server utilise les groupes de fichiers. La gestion des données FILESTREAM n'échappe pas à la règle, il est donc nécessaire de définir une groupe de fichiers. La particularité de ce groupe de fichiers vient du fait qu'il n'est pas composé de fichiers mais qu'il correspond à un dossier créé préalablement sur le système de fichiers.

*Exemple*

*Dans l'exemple suivant le groupe de fichiers FilestreamGroup est défini et le dossier C:\gescom\images est associé au groupe de fichiers. Le dossier c:\gescom doit être créé sur le système de fichiers, l'instruction Transact SQL se charge de définir le sous-dossier images de la même façon qu'elle se charge de définir le fichier de données sur un groupe de fichiers classique.*




Une colonne de type FILESTREAM peut alors être définie, il est toutefois possible de définir une colonne de ce type uniquement dans les tables possédant une colonne de type uniqueidentifier rowguidcol, n'acceptant pas les valeurs null et avec une contrainte d'unicité ou bien de clé primaire.

*Exemple*

*La table catalogue est donc définie à cet effet :*

```
create table catalogue (
   id uniqueidentifier rowguidcol not null,
   article nvarchar(16),
   image_art varbinary(max) filestream,
   constraint pk_catalogue primary key(id)
);
```

```
Messages
Commande(s) réussie(s).
```

Il est maintenant possible d'ajouter de nouvelles lignes d'informations dans la table. Comme le Transact SQL n'est pas le langage le mieux adapté pour importer des images (il faut lui préférer .Net), c'est une chaîne de caractères qui sera considérée comme une donnée binaire.

```
insert into catalogue (id, article, image_art) values
   (NEWID(),'000397', CAST('image article 000397' as varbinary(max))),
   (NEWID(),'000432', CAST('image article 000432' as varbinary(max)));
```

```
Messages
(2 ligne(s) affectée(s))
```

Il est possible de consulter le contenu du dossier cible et de constater qu'il contient maintenant des informations.

```
C:\WINDOWS\system32\cmd.exe

C:\gescom\images>dir
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est 9CC4-80FE

 Répertoire de C:\gescom\images

28/06/2009  22:37    <REP>          .
28/06/2009  22:37    <REP>          ..
28/06/2009  22:33    <REP>          $FSLOG
28/06/2009  22:37    <REP>          972fce65-cb2a-4dc6-866c-f4e1fd52181d
28/06/2009  22:33               422 filestream.hdr
               1 fichier(s)              422 octets
               4 Rép(s)   9 555 251 200 octets libres

C:\gescom\images>
```

Le fichier filestram.hbr ne doit surtout pas être supprimé.

# Les données spatiales

Les applications qui travaillent avec des données géographiques sont maintenant nombreuses et elles permettent un repérage plus rapide de l'information, souvent avec comme objectif de dresser un itinéraire. Mais les applications peuvent également utiliser les données géographiques pour obtenir une représentation visuelle des données ou bien pour faire une analyse géographique des données (où sont répartis nos principaux clients ? …).

Toutefois, il n'est pas possible de gérer de la même façon les données relatives à un schéma d'une ville ou d'un quartier et celles relatives à un pays. En effet, le chemin pour se rendre de la place de la Concorde à la place de l'Étoile à Paris est infiniment plus court que le trajet à effectuer pour se rendre de Paris à Mayotte. Dans le premier cas il est possible de considérer que la terre est plate alors qu'il n'est plus possible de faire cette approximation dans le second cas.

Pour répondre à ces contraintes, SQL Server propose les types geometry et geography. Le type geometry travaille sur un plan à deux dimensions et permet la représentation des données à une échelle locale. Le type geography utilise la latitude et la longitude pour stocker les différentes informations.

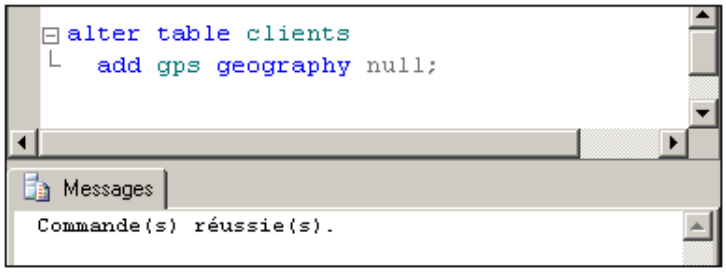
Pour être en mesure de travailler avec les principaux outils de représentation cartographique du marché, SQL Server respecte les spécifications émises par l'Open Geospatial Consortium (OGC). Il propose les méthodes et les propriétés du type geometry conformément aux recommandations de l'OGC.

Ces types de données (geometry et geography) sont définis comme étant des types CLR.

Il est possible de définir un index sur les colonnes qui hébergent des données géographiques par l'intermédiaire d'une grille à multiples niveaux.

*Exemple*

*La table clients est modifiée pour y ajouter une colonne de type geography :*



# Introduction

La gestion des sauvegardes reste une des tâches les plus importantes qui doit être réalisée par l'administrateur de bases de données. Si les opérations de sauvegardes sont planifiées avec exactitude et rigueur, il est tout à fait envisageable de réaliser les sauvegardes sous forme de travaux automatisés et le responsable sera prévenu par e-mail du bon déroulement des opérations.

Les sauvegardes sont réalisées pour se prémunir des pertes de données suite à :

- une panne de support,
- des erreurs utilisateur,
- une perte permanente du serveur.

Les sauvegardes SQL Server permettent de sauvegarder la base de données alors que des utilisateurs y sont connectés. Cette sauvegarde va prendre en compte tous les fichiers constituant la base de données et va enregistrer leur emplacement. Le processus de sauvegarde assure la cohérence des données et des journaux en capturant toutes les activités survenant durant le processus de sauvegarde.

Bien que la base reste accessible durant la sauvegarde, certaines opérations sont impossibles, à savoir :

- la création ou la modification d'une base de données (notamment l'extension automatique du journal des transactions),
- la création d'un index,
- l'exécution d'opérations non journalisées car le processus de sauvegarde utilise le journal pour garantir la cohérence des données.

# Planification

La planification des opérations de sauvegarde entraîne également celle de la restauration, en effet l'une ne va pas sans l'autre. Ce sont les exigences en matière de disponibilité des données qui vont constituer le critère déterminant pour la mise en place des sauvegardes. Pour réaliser cette planification il faut réfléchir à tous les incidents qui peuvent survenir et connaître quels seront les besoins en sauvegarde pour restaurer au mieux l'information. Il sera ensuite possible d'automatiser tous ces travaux de sauvegarde, et des tests de restauration valideront le bon déroulement des procédures de sauvegarde.

## 1. Les questions

Les principales questions qu'il faut se poser pour planifier au mieux les sauvegardes sont :

- Quelle est la taille de chacune des bases de données ?
- Quel est le volume des modifications de données ?
- Certaines tables sont-elles plus sujettes que d'autres aux modifications ?
- Combien de temps la base de données peut-elle rester indisponible ?
- La perte de modifications est-elle cruciale ?
- Est-il facile de recréer les données perdues ?
- Quelles sont les périodes importantes d'utilisation de la base de données ?
- La base subit-elle des surcharges de travail ponctuelles pendant lesquelles il n'est pas envisageable de lancer une sauvegarde ?
- Les utilisateurs doivent-ils continuer à accéder aux données pendant les opérations de sauvegarde ?
- Quel est le laps de temps entre les troncatures (suppression de la partie inactive) du journal des transactions ?
- Les sauvegardes sont-elles conservées de façon cyclique ?
- Le serveur SQL est-il en cluster ?
- Le serveur SQL est-il dans un environnement multiserveur avec une administration centralisée ?

La durée des opérations de sauvegarde va dépendre principalement du support sur lequel les sauvegardes sont effectuées. Deux supports sont possibles : les bandes (à condition que le lecteur de bande soit installé sur le serveur SQL) et les fichiers.

## 2. Choisir une stratégie de sauvegarde

Pour chaque base de données, il faut choisir une stratégie de sauvegarde qui va être une combinaison de sauvegardes complètes de bases de données et de sauvegardes du journal des transactions. Pour améliorer les performances de sauvegardes, SQL Server propose des sauvegardes différentielles. De plus, toutes ces opérations peuvent être réalisées sur la totalité de la base ou sur un groupe de fichiers uniquement.

Une bonne connaissance des différentes méthodes de sauvegarde, permet de mettre en œuvre le plan de sauvegarde qui répond au mieux face aux exigences constatées lors de la planification.

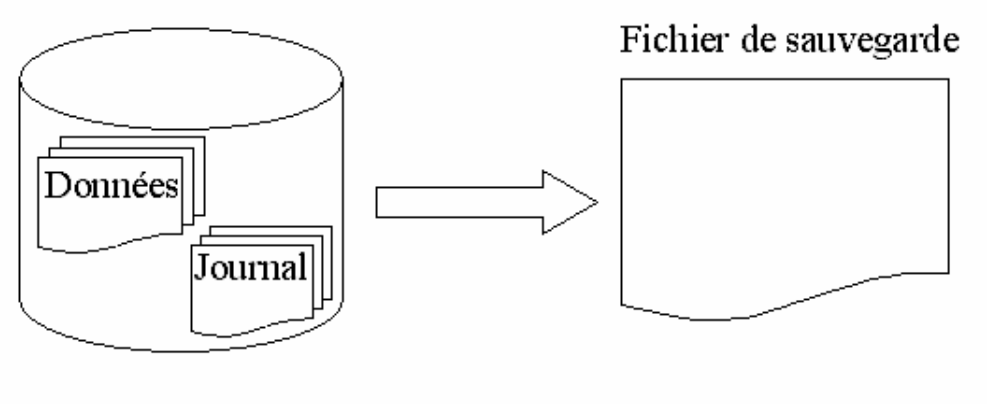
---

La stratégie de sauvegarde doit être fixée base par base en fonction des besoins et des évolutions de chacune d'entre elles.

### a. Sauvegarde d'une base de données

La sauvegarde totale d'une base de données permet de fournir un point de départ pour les restaurations. Si uniquement des sauvegardes complètes de base de données sont effectuées, en cas de problème, les transactions validées depuis la dernière sauvegarde complète seront perdues.

Les sauvegardes complètes nécessitent un temps relativement long et occupent sur le support de sauvegarde un espace conséquent. C'est pourquoi, même si elles constituent un point de départ obligatoire à toute stratégie de sauvegarde, les sauvegardes complètes de base de données restent bien adaptées aux bases de faibles volumes et pour lesquelles il est possible de reproduire facilement toutes les transactions qui ont eu lieu depuis la dernière sauvegarde complète.

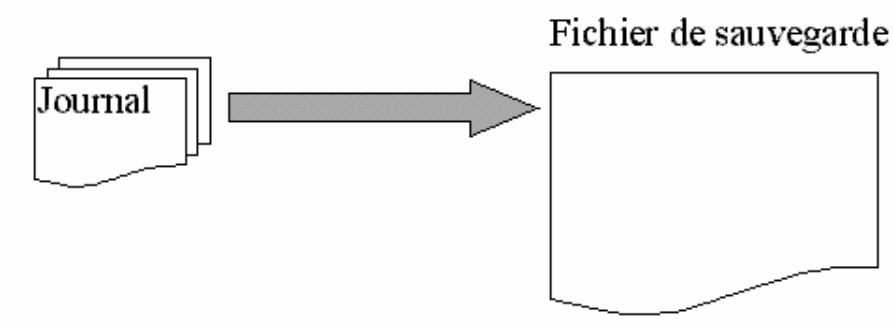



### b. Sauvegarde du journal des transactions

En complément des sauvegardes complètes, il est toujours possible de mettre en place une politique de sauvegarde des journaux de transactions. La sauvegarde des journaux présente deux avantages majeurs.

Il est possible de récupérer la totalité ou une grande partie des transactions validées depuis la dernière sauvegarde complète de la base.

La taille du fichier journal ne risque pas de grandir de façon anarchique car lors de chaque sauvegarde du journal, il est possible de demander de le tronquer. Le risque de saturer le disque suite à l'extension du fichier journal est ainsi diminué. La sauvegarde des journaux peut être réalisée par un travail qui est planifié pour une exécution régulière.




SQL Server génère des points de contrôle synchronisation (CHECKPOINT) automatiques. Le but de ces points de synchronisation est de stocker sur disque l'ensemble des transactions validées pour lesquelles les modifications sont encore en mémoire. Ainsi en cas de restauration automatique suite à un arrêt brutal du serveur le volume des données à restaurer sera faible car il ne concernera que les données appartenant à des transactions validées depuis le dernier point de synchronisation.

Le point de synchronisation peut être déclenché ponctuellement par l'intermédiaire de l'instruction Transact SQL CHECKPOINT.

**Syntaxe**

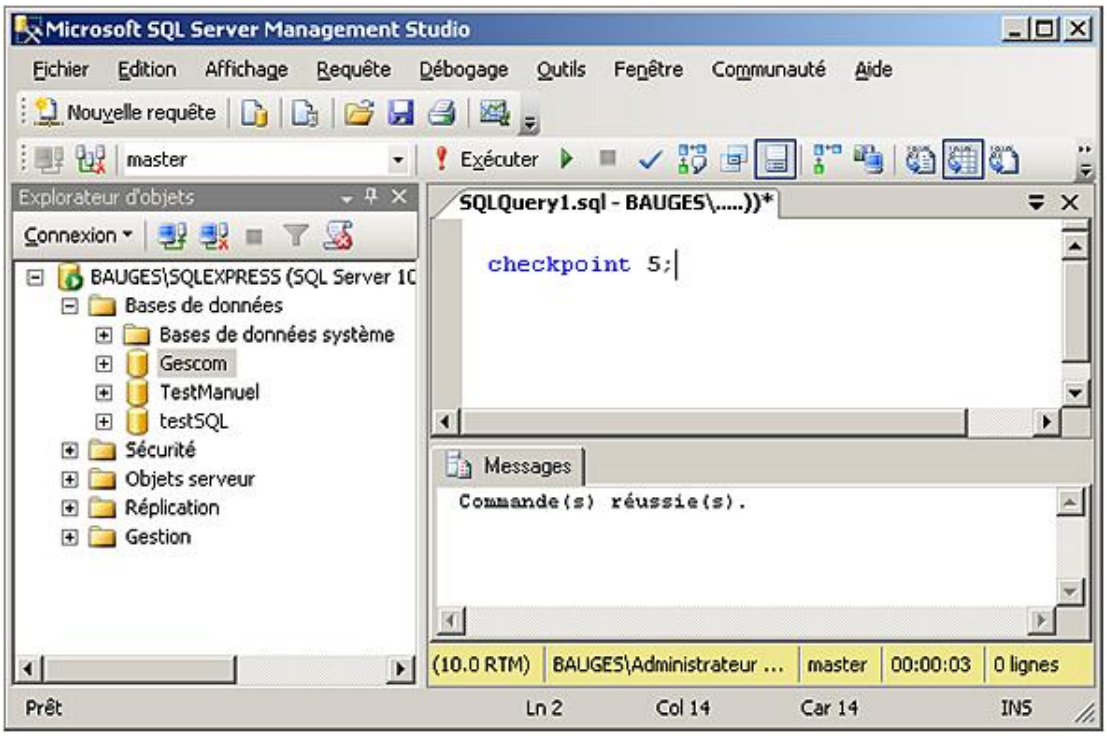
```
CHECKPOINT [valeur][;]
```

`Valeur`

Précise le nombre de secondes dont dispose SQL Server pour terminer le point de synchronisation.

*Exemple*

*Dans l'exemple suivant le point de synchronisation est réalisé dans les 5 secondes.*

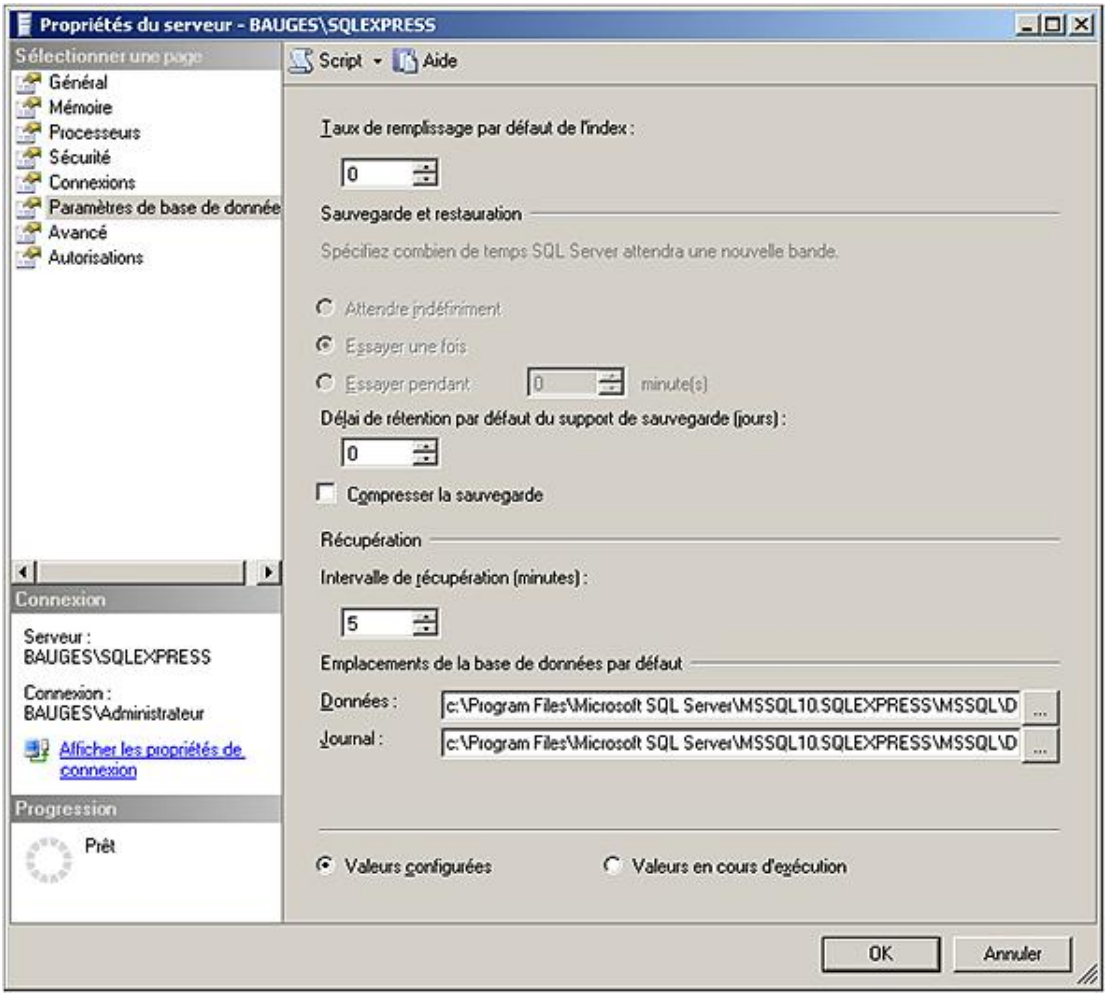



Le point de synchronisation peut également être réalisé de façon automatique en fonction de certains critères :

- le respect des paramètres fixés à l'aide de l'option recovery interval,
- le journal est rempli à plus de 70 % alors que la base est configurée pour tronquer le journal lors de l'exécution des points de synchronisation,
- le moteur de base de données est arrêté, sauf si cet arrêt est demandé avec l'instruction SHUTDOWN WITH NOWAIT.

L'option **RECOVERY INTERVAL** permet de définir au niveau du serveur le nombre maximal de minutes que peut prendre une restauration automatique de la base depuis le dernier point de synchronisation. La fréquence des points de synchronisation est fixée par SQL Server en fonction de l'activité de mise à jour sur la base. Par défaut, la valeur de ce paramètre est 0, ce qui signifie que SQL Server gère automatiquement la fréquence des points de synchronisation. Il est recommandé de conserver cette valeur. Cependant, si les points de synchronisation sont mal adaptés par rapport à l'activité du serveur, il est possible de modifier la valeur de ce paramètre soit avec la procédure **sp_configure**, soit depuis la fenêtre des propriétés du serveur depuis SQL Server Management Studio.

Comme l'illustre l'écran suivant, l'intervalle de récupération est fixé sur la page **Paramètres de base de données**.

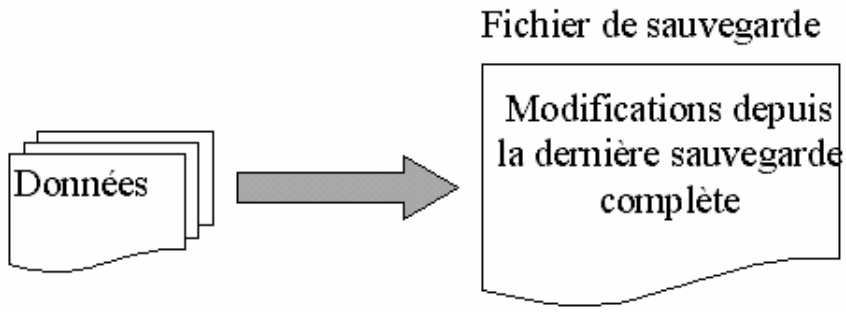



Si la base de données utilise le mode de récupération simple, le journal n'est utilisé que lors des récupérations automatiques, il est donc inutile de conserver les informations sur les transactions qui ont pris fin avant le dernier point de synchronisation. Pour éviter que la taille du journal croisse de façon illimitée, SQL Server élimine la partie inactive du journal lors de chaque point de synchronisation.

### c. Les sauvegardes différentielles

Si les sauvegardes du journal sont des opérations lourdes car le volume des journaux peut rapidement devenir très important, une alternative intéressante peut être mise en place avec les sauvegardes différentielles. Les sauvegardes différentielles ne vont prendre en compte que les données modifiées depuis la dernière sauvegarde complète.

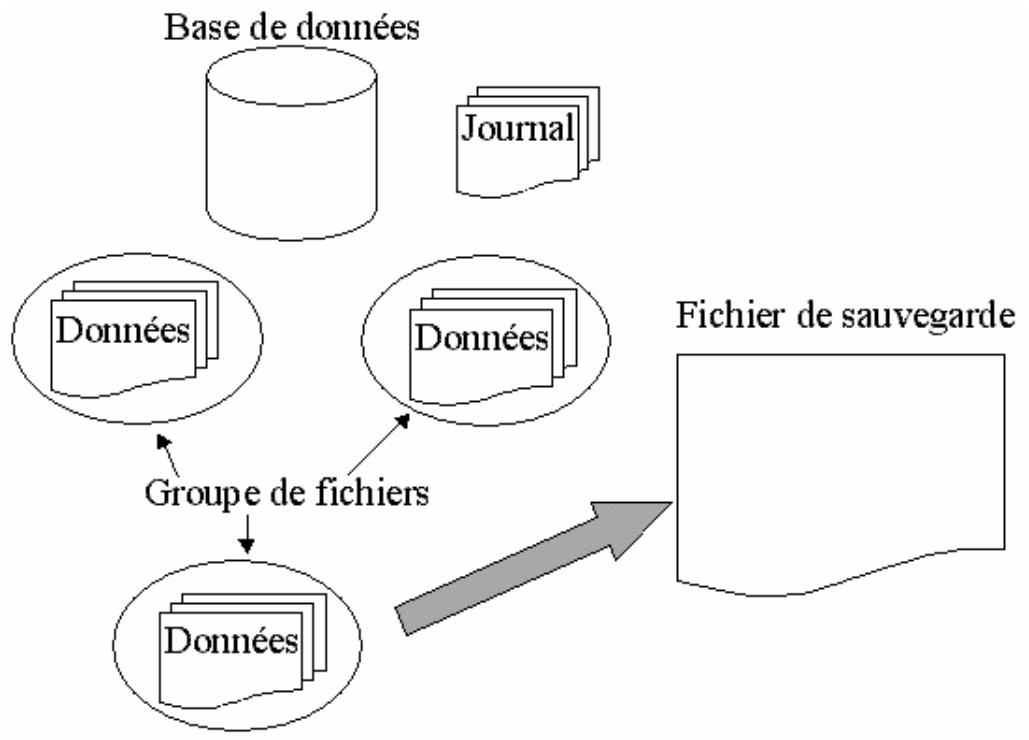
Les sauvegardes différentielles sont plus rapides et moins volumineuses que les sauvegardes complètes, et associées aux sauvegardes du journal des transactions, elles peuvent constituer une solution de sauvegarde à la fois rapide et performante.




### d. Les sauvegardes par groupe de fichiers

Si la base de données représente un volume important de données, les sauvegardes complètes et différentielles

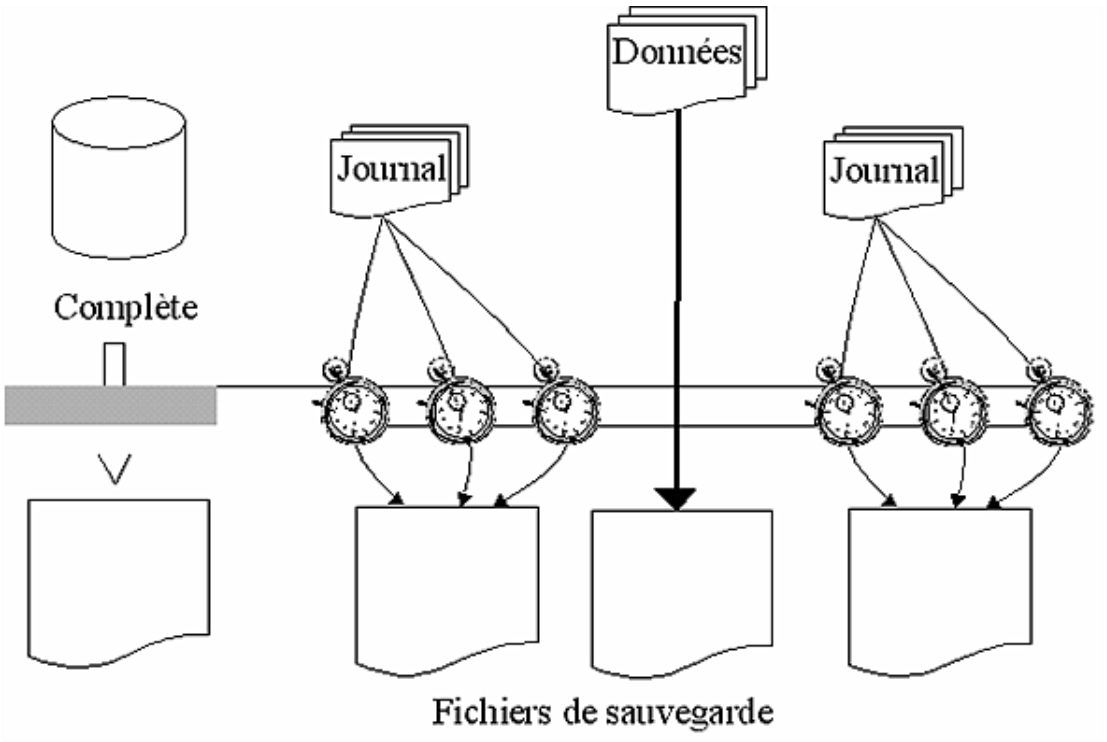
peuvent demander un temps d'exécution très long. Pour réduire ce temps, il est possible de sauvegarder les données par groupes de fichier. Une telle opération est bien sûr possible si lors de la création de la base, des groupes de fichiers ont été définis.




### e. Les combinaisons possibles

La bonne solution pour réaliser des sauvegardes passe par une combinaison des différentes méthodes de sauvegarde.

Par exemple, il est possible de combiner les sauvegardes complètes, les sauvegardes du journal des transactions et les sauvegardes différentielles pour minimiser à la fois le volume et le temps des sauvegardes, et la perte des transactions validées. En mettant en place ces opérations sous forme de travaux planifiés avec notification en fin d'exécution, les opérations de sauvegardes peuvent se dérouler automatiquement sans effort de la part de l'administrateur.



Si une base s'étend sur plusieurs groupes de fichiers (primary, données...), il est alors possible de réaliser les sauvegardes par groupes de fichiers. Pour chaque groupe de fichiers, il faut appliquer une stratégie de sauvegarde (complète, différentielle et journaux de transactions). Il n'est pas nécessaire d'effectuer simultanément le même type de sauvegarde sur tous les groupes de fichiers.

Dans la combinaison présentée ci-dessus, les trois types de sauvegarde sont utilisés. La sauvegarde complète constitue le point de départ obligatoire. Par la suite, les journaux sont sauvegardés régulièrement afin de perdre le minimum de données. Pour minimiser les temps de restauration, une sauvegarde différentielle est effectuée afin d'y conserver toutes les modifications qui sont intervenues depuis la dernière sauvegarde complète.

Toutes les stratégies de sauvegarde commencent toujours par une sauvegarde complète de la base.

# La mise en œuvre des sauvegardes

Les opérations de sauvegarde peuvent être mises en place soit à partir de SQL Server Management Studio, soit à l'aide de procédures stockées en Transact SQL. Comme toujours, la solution graphique propose la facilité de mise en œuvre mais les commandes Transact SQL permettent quant à elles d'accéder à la totalité des options proposées.

## 1. Les modes de récupération

Les possibilités offertes au niveau de la sauvegarde et donc de la restauration sont directement liées au mode de configuration défini au niveau de chaque base de données. Les trois modes de récupération qu'il est possible de configurer sont :

- **le mode de récupération simple** : le journal est utilisé uniquement pour garantir la persistance des opérations apportées sur les données. Il est tronqué lors de chaque point de synchronisation.

- **le mode de récupération complet** : dans ce mode, toutes les transactions sont consignées dans le journal et y restent enregistrées même après le point de synchronisation. En cas d'échec, la perte de données est réduite car l'opération de restauration est possible jusqu'au point de défaillance (sous réserve d'avoir adopté la bonne politique de sauvegarde). Il s'agit du mode de récupération par défaut défini au niveau des bases de données utilisateur.

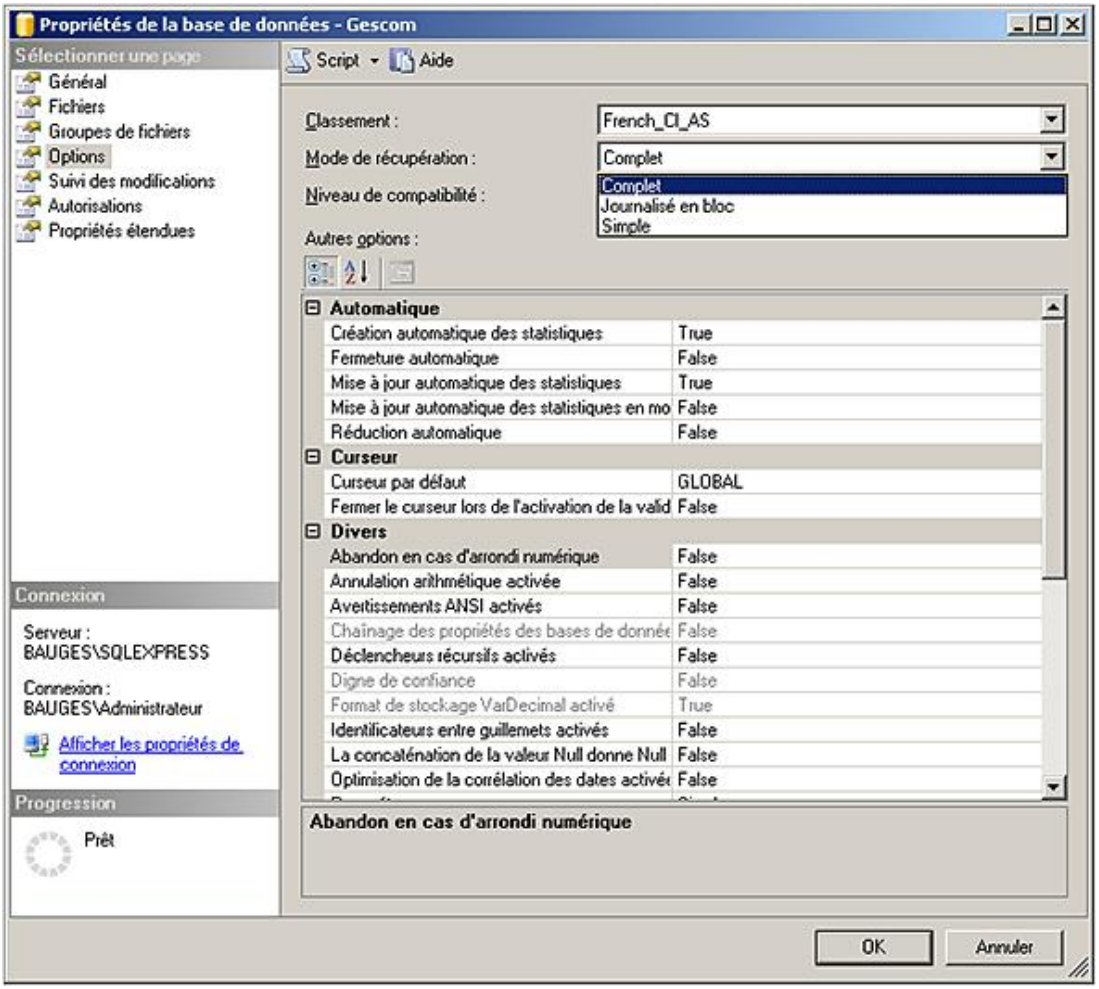
- **le mode de récupération journalisé en bloc** : dans ce mode de récupération avancé, non seulement les informations relatives aux transactions sont enregistrées dans le journal, mais également certaines opérations affectant les données, comme la création d'index.

Pour configurer le mode de récupération, il est possible d'exécuter l'instruction ALTER DATABASE.

**Syntaxe**

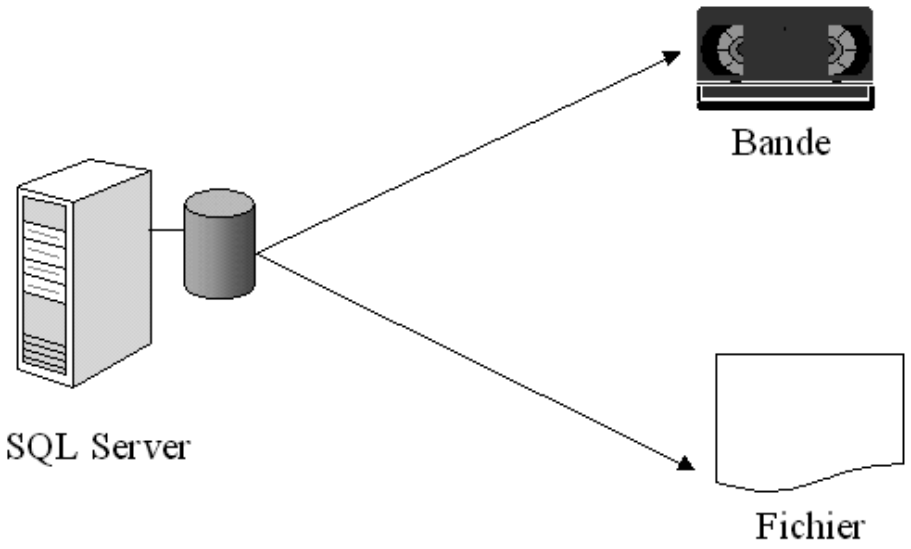
```
ALTER DATABASE nomBaseDeDonnées
SET RECOVERY { FULL | BULK_LOGGED | SIMPLE }
```

Il est également possible de modifier les propriétés de la base depuis SQL Server Management Studio. Ce dernier cas est illustré ci-dessous pour configurer la base Gescom en mode de récupération complet.




## 2. La destination des sauvegardes

Les sauvegardes peuvent être dirigées sur différents supports : bande, disque dur.




### a. Disque dur

Les sauvegardes sur disques sont réalisées à l'intérieur de fichier du système d'exploitation. La sauvegarde peut être réalisée sur un disque local ou sur un disque distant. Pour que SQL Server puisse accéder à un partage réseau, celui-ci doit être mappé à un lecteur réseau au niveau de l'utilisateur Windows dans le contexte duquel, le service

s'exécute. Il est possible d'utiliser la procédure **xp_cmdshell** pour définir ce lecteur réseau.

Il est préférable de réaliser les sauvegardes sur un disque autre que celui qui contient les fichiers journaux et de données de la base afin de se prémunir des pannes disque.

Lorsque la sauvegarde sur fichier est terminée, il est prudent de sauvegarder ce fichier sur un support amovible (bande, disquette de type zip...) afin de stocker les sauvegardes dans un lieu distinct de celui où est le serveur.

### b. Bandes

Les bandes représentent un moyen économique et efficace pour conserver les sauvegardes. De plus elles possèdent un volume important de stockage. Il est facile de les conserver en dehors du site de production pour une meilleure qualité des sauvegardes.

La gestion des bandes pour les sauvegardes n'est envisageable que si le lecteur de bande est local au serveur SQL.

Lors d'une sauvegarde sur bande, SQL Server enregistre automatiquement : le nom de la base de données, la date, l'heure et le type de sauvegarde.

Il existe des instructions Transact SQL :

- UNLOAD : rembobine et éjecte la bande.
- NOUNLOAD : pas de rembobinage et pas d'éjection.
- BLOCKSIZE : taille des blocs physiques en octets.
- FORMAT : écrit un en-tête dans les fichiers utilisés.
- SKIP : ignorer les étiquettes de bande ANSI.
- NOSKIP : lire les étiquettes de bande ANSI.
- RESTART : reprendre l'opération de sauvegarde à partir du point d'interruption.
- REWIND : rembobine et libère la bande.
- NOREWIND : SQL conserve la bande après la fin de l'opération de sauvegarde.

Les bandes peuvent contenir aussi bien des sauvegardes SQL Server que des sauvegardes Windows.

## 3. Les principaux paramètres

La mise en place des sauvegardes peut être réalisée soit par des procédures stockées, soit par SQL Server Management Studio. Lorsque les étapes constituant une opération de sauvegarde sont validées, il est possible de les réunir pour constituer un travail planifié et donc géré par l'Agent SQL Server.

### a. Les permissions

Pour réaliser une opération de sauvegarde, l'utilisateur doit posséder certains droits. Trois rôles prédéfinis contiennent les autorisations suffisantes pour sauvegarder une base de données. Il est bien sûr possible de définir ses propres rôles et d'y accorder les autorisations nécessaires.

Les utilisateurs membres d'un des rôles suivants sont capables de réaliser une sauvegarde de base de données :

- rôle de serveur **sysadmin**,
- rôle de base de données **db_owner**,
- rôle de base de données **db_backupoperator**.

### b. La sauvegarde des bases de données système

La base de données système qui contient toutes les informations relatives au bon fonctionnement du serveur est la base **Master**. Cette base doit être sauvegardée après chaque modification, surtout après la création de base de données utilisateur. En effet si la base n'est pas référencée dans **Master**, il est impossible d'y accéder. Il ne faut pas oublier non plus que la base **Master** contient la définition de toutes les connexions, ainsi que toutes les références vers les serveurs liés.

Pour reconstruire les bases de données système, il faut passer par le programme d'installation et choisir l'option REBUILDDATABASE.

La base **MODEL** sert de base de départ pour toutes les bases de données utilisateur.

Ces bases de données système sont à sauvegarder après chaque modification. En période de fonctionnement, les modifications apportées sur ces bases sont normalement assez rares et il n'est donc pas utile de les sauvegarder si la base n'a pas évolué. Il est ainsi possible de réduire les temps de sauvegarde.

### c. La sauvegarde des bases de données utilisateur

Ce sont les bases qui sont le plus concernées par les sauvegardes car elles contiennent toutes les informations de l'entreprise. Selon l'importance des données qui y sont stockées et le nombre de modifications qui y sont apportées un plan de sauvegarde va être déployé. Il est tout de même intéressant de noter quelques étapes qui nécessitent une sauvegarde totale de la base :

- Après la création de la base.
- Après la création d'un index : en effet le journal des transactions mémorise la création de l'index, mais parfois la création d'un index peut demander plus de temps que de restaurer la totalité de la base.
- Après la suppression du contenu du journal des transactions, une sauvegarde totale permet de ne pas perdre de données.
- Après l'exécution d'une commande non journalisée.

### d. Les fichiers de sauvegarde

SQL Server ne travaille pas avec la notion de fichiers de sauvegarde, mais plus exactement des unités de sauvegarde. Il existe deux catégories d'unité de sauvegarde :

- les unités de sauvegarde logique,
- les unités de sauvegarde physique.

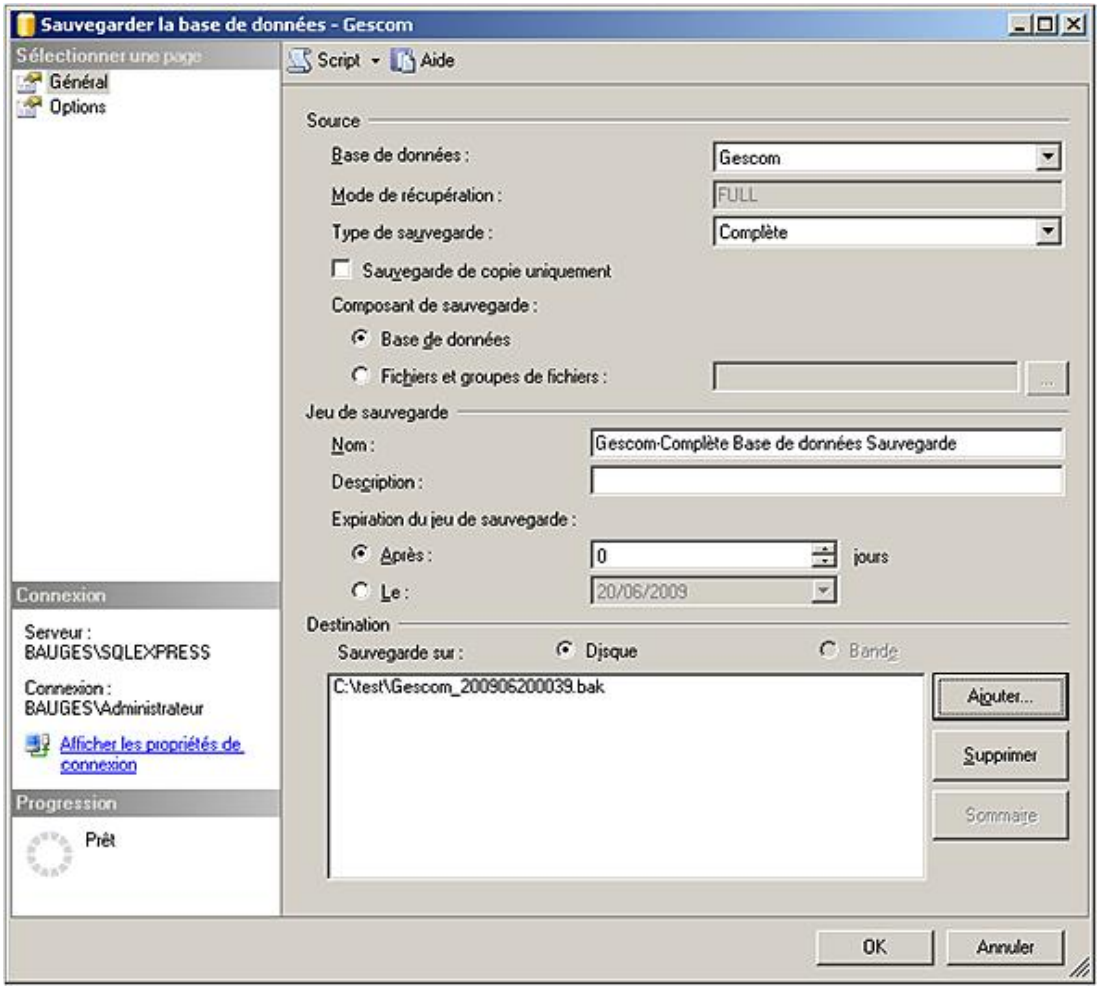
La distinction entre fichier et unité de sauvegarde provient du fait qu'une unité de sauvegarde peut être composée de plusieurs fichiers.

Une unité de sauvegarde peut contenir plusieurs sauvegardes d'une ou de plusieurs bases de données.

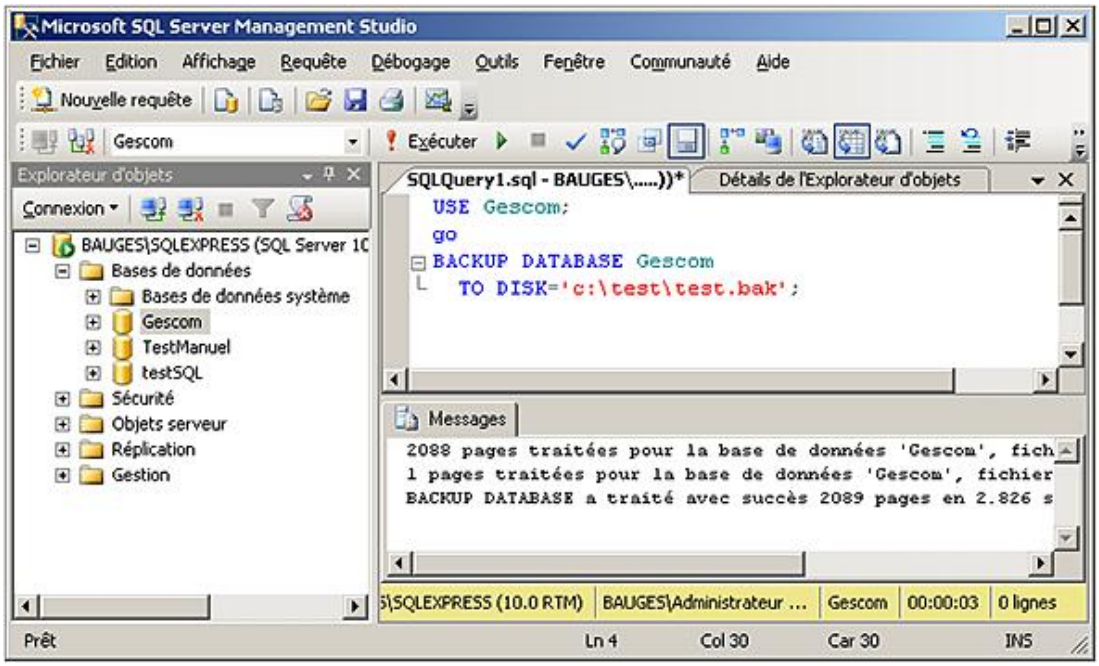
#### Les unités physiques

Une unité de sauvegarde physique correspond au nom complet du fichier sous Windows. Une unité de sauvegarde physique correspond le plus souvent à une utilisation ponctuelle d'un support de sauvegarde. Par exemple avant de réaliser une opération sensible sur la base, une sauvegarde complète de la base peut être réalisée dans une unité de sauvegarde physique.

L'unité physique est référencée directement dans les options de l'instruction BACKUP, ou bien le nom de l'unité physique est précisé sur la fenêtre **Sauvegarder la base de données** de SQL Server Management Studio.




En Transact SQL, l'unité physique est référencée directement par l'instruction BACKUP.

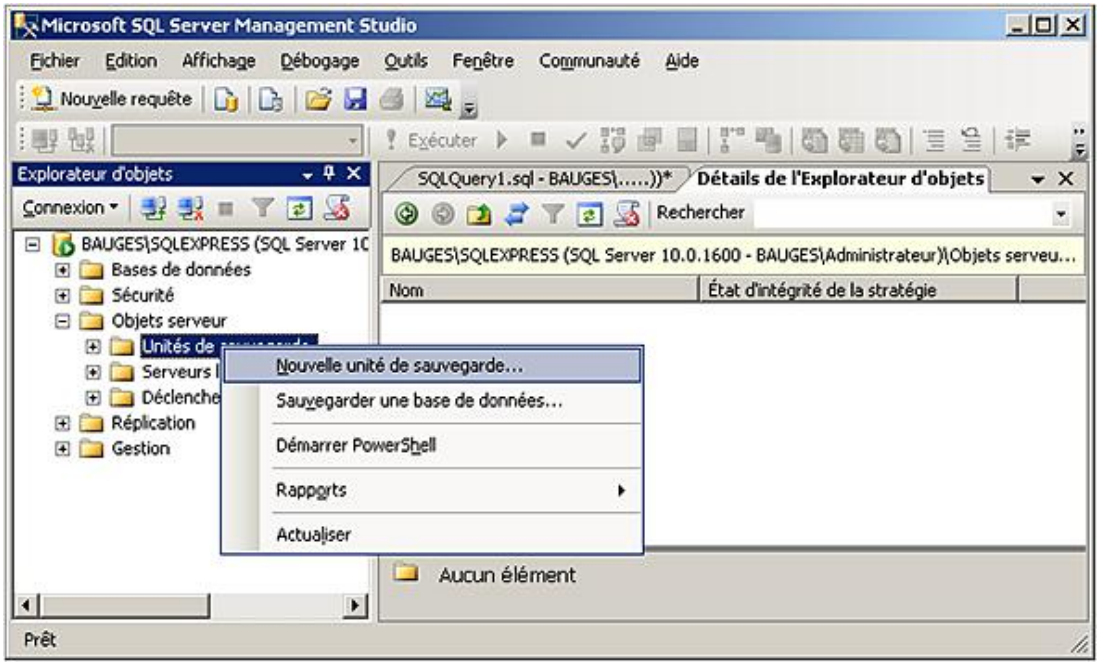



Si la durée de validité de la sauvegarde, que ce soit en nombre de jours ou en date de fin de validité, n'est pas spécifiée, alors SQL Server utilise la valeur définie au niveau du serveur par l'intermédiaire du paramètre de configuration **media retention**. Cette propriété est une propriété avancée et n'est donc accessible qu'après l'exécution de l'instruction show advanced option.

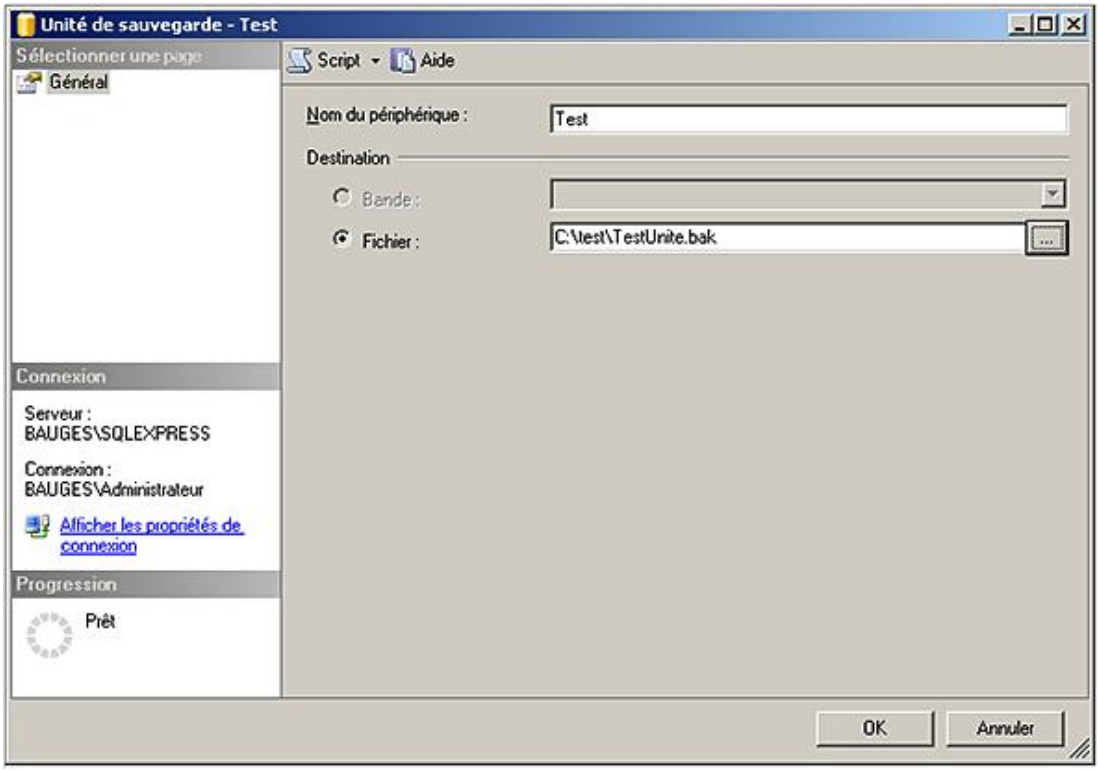
## Les unités logiques

Derrière une unité de sauvegarde logique, il se cache une unité physique. Les unités de sauvegarde logique permettent de référencer logiquement les différents supports de sauvegarde qui peuvent être utilisés au niveau de la base. Par exemple, dans une politique de sauvegarde qui contient une sauvegarde complète chaque jour et une sauvegarde différentielle toutes les 4 heures, il est possible de définir les unités de sauvegarde completLundi, diffLundi, completMardi, diffMardi... de façon à utiliser des unités différentes chaque jour de la semaine. Les opérations de sauvegarde utilisent un nom logique, c'est-à-dire qu'il ne leur est pas nécessaire de connaître l'emplacement physique des fichiers.

Depuis SQL Server Management Studio, il est possible de créer une unité logique de sauvegarde en sélectionnant l'option **Nouvelle unité de sauvegarde** depuis le menu contextuel associé au nœud **Objets serveur - Unités de sauvegardes** de l'explorateur d'objets.



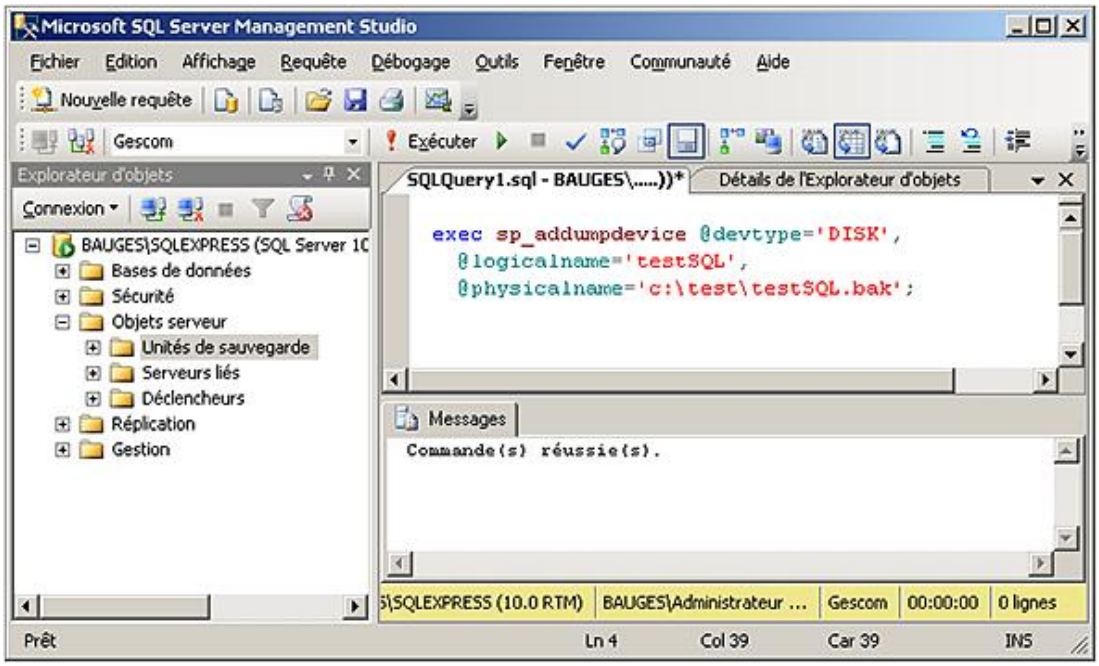

Dans l'exemple suivant, l'unité **Test** est créée en s'appuyant sur un fichier physique.




La fenêtre des propriétés de l'unité de sauvegarde permet de connaître le contenu d'une unité.

Le même type d'opération peut être réalisé en Transact SQL.

C'est la procédure **sp_addumpdevice** qui permet de définir de nouvelles unités logiques.

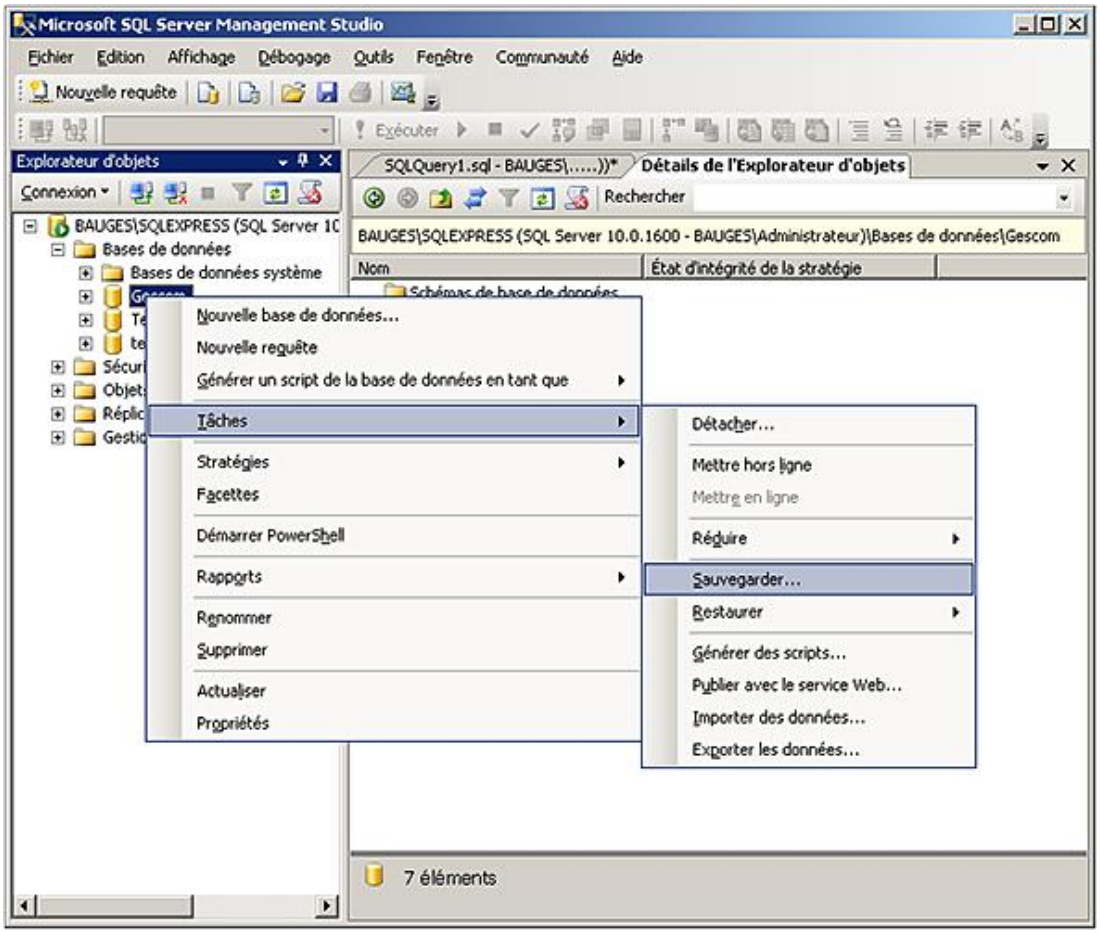



Les unités logiques permettent une gestion simplifiée des sauvegardes au travers de SQL Server Management Studio. Elles permettent également de réutiliser plus efficacement l'espace disque des anciennes sauvegardes, tout en facilitant la mise en place de procédures automatiques sur les opérations de sauvegarde.

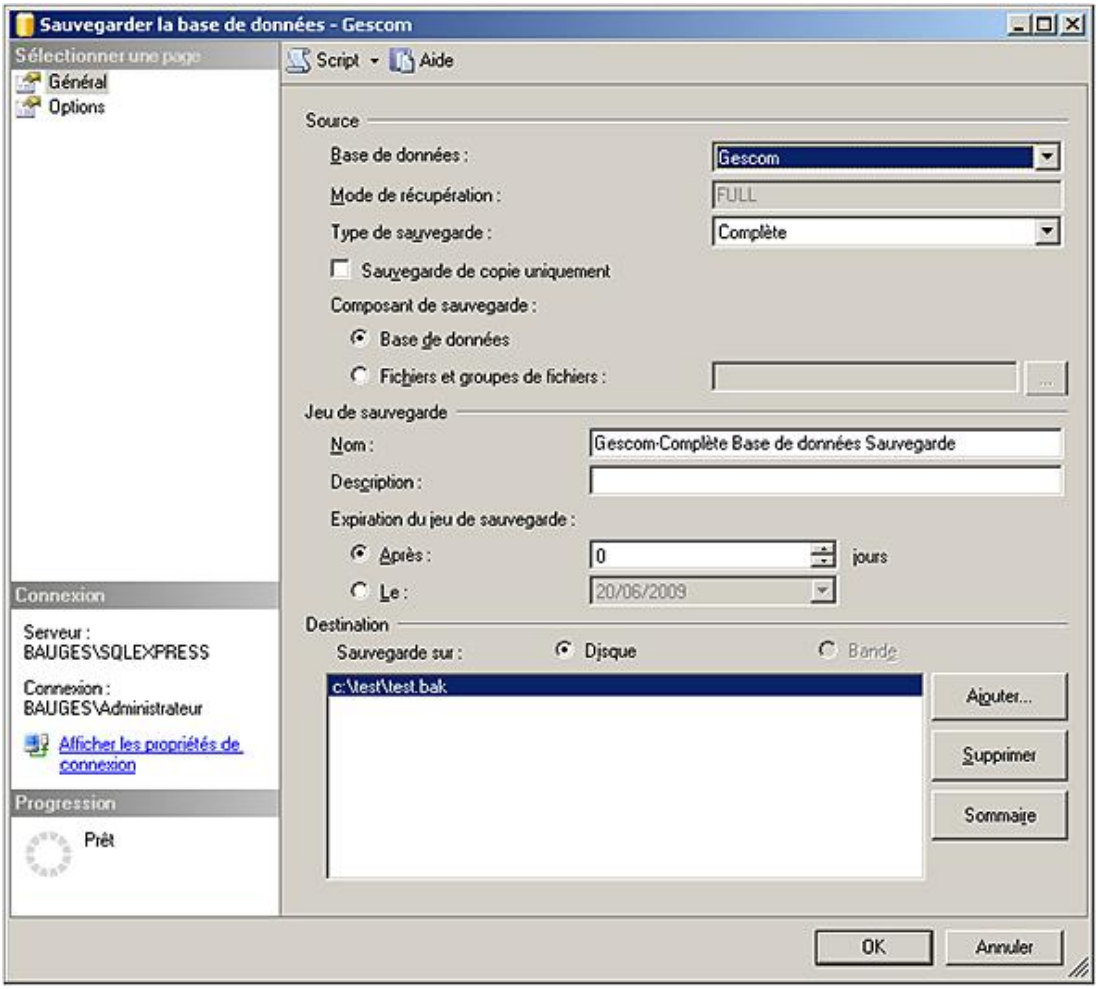
## 4. L'instruction BACKUP

L'instruction BACKUP permet de sauvegarder aussi bien la base de données que le journal de transaction. Les opérations de sauvegarde peuvent être réalisées en Transact SQL par l'intermédiaire de cette instruction, mais il est possible de passer par l'interface graphique de SQL Server Management Studio.

L'écran ci-dessous illustre comment demander l'exécution d'une sauvegarde depuis SQL Server Management Studio.




La fenêtre de création d'une nouvelle sauvegarde permet de définir toutes les options relatives à l'exécution de cette sauvegarde.




Toutes les options de l'opération de sauvegarde sont disponibles par l'intermédiaire de la page **Options**.

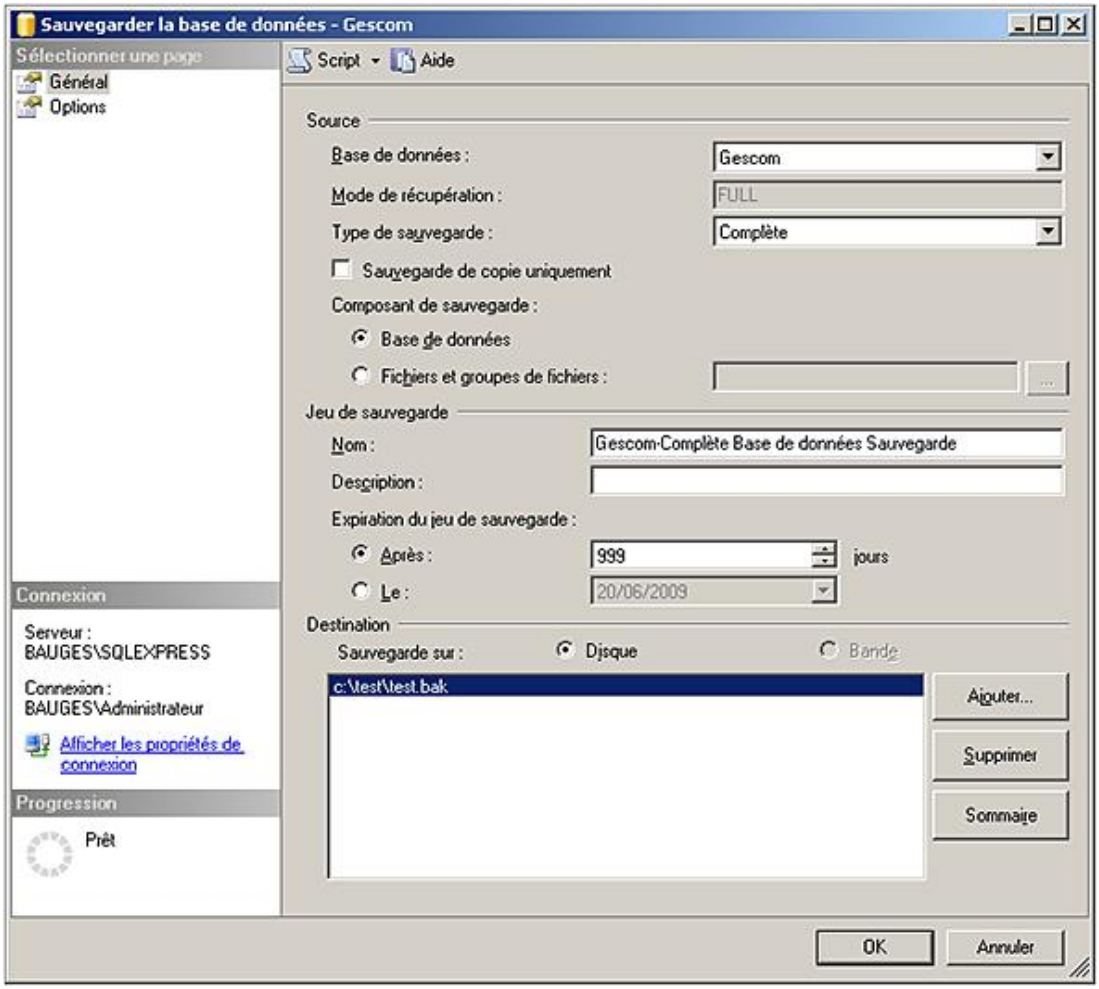
Dans le cadre de la réutilisation d'un fichier existant, il est possible, soit d'ajouter la sauvegarde à celles déjà contenues dans le fichier, soit au contraire d'effacer toutes les données présentes dans le fichier.

Dans le cas où l'instruction BACKUP est utilisée directement, la réutilisation d'un fichier entraîne trois options possibles :

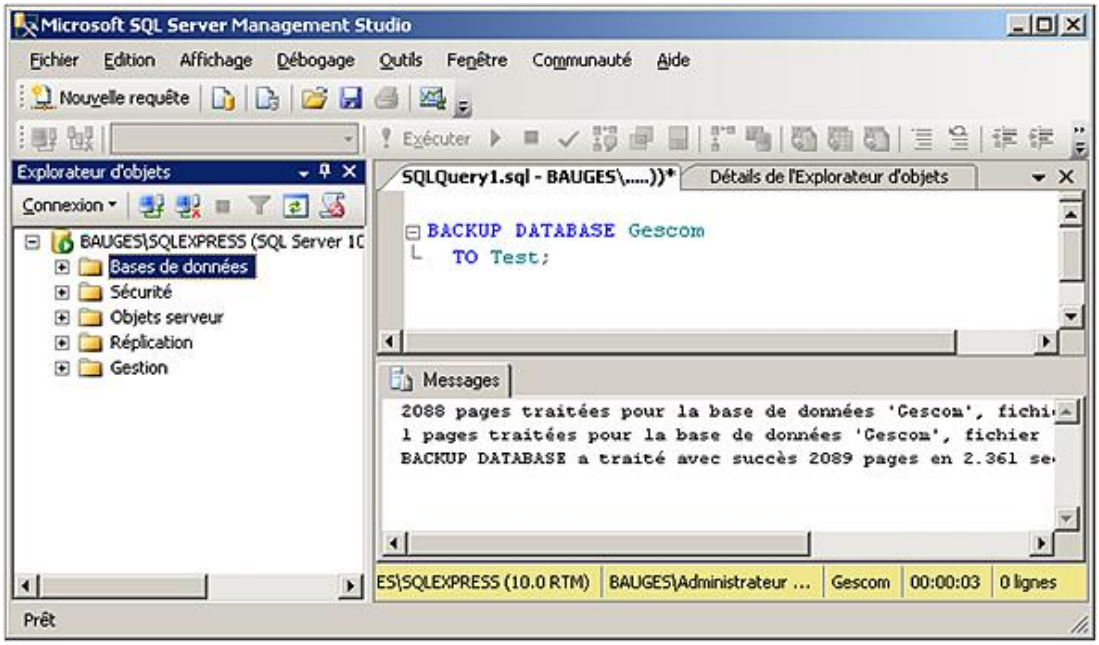
- INIT : pour remplacer le contenu d'un fichier permanent, à condition que la date d'expiration de la sauvegarde soit dépassée (option EXPIRATE) et que le fichier ne soit pas membre d'un jeu de sauvegarde.
- NOINIT : pour ajouter la sauvegarde à celles déjà présentes dans le fichier.
- FORMAT : pour pouvoir réutiliser un fichier qui a participé à une sauvegarde sur plusieurs fichiers.

### a. Sauvegarde complète

C'est le point de départ pour toute stratégie de sauvegarde.




*Demande de sauvegarde complète*

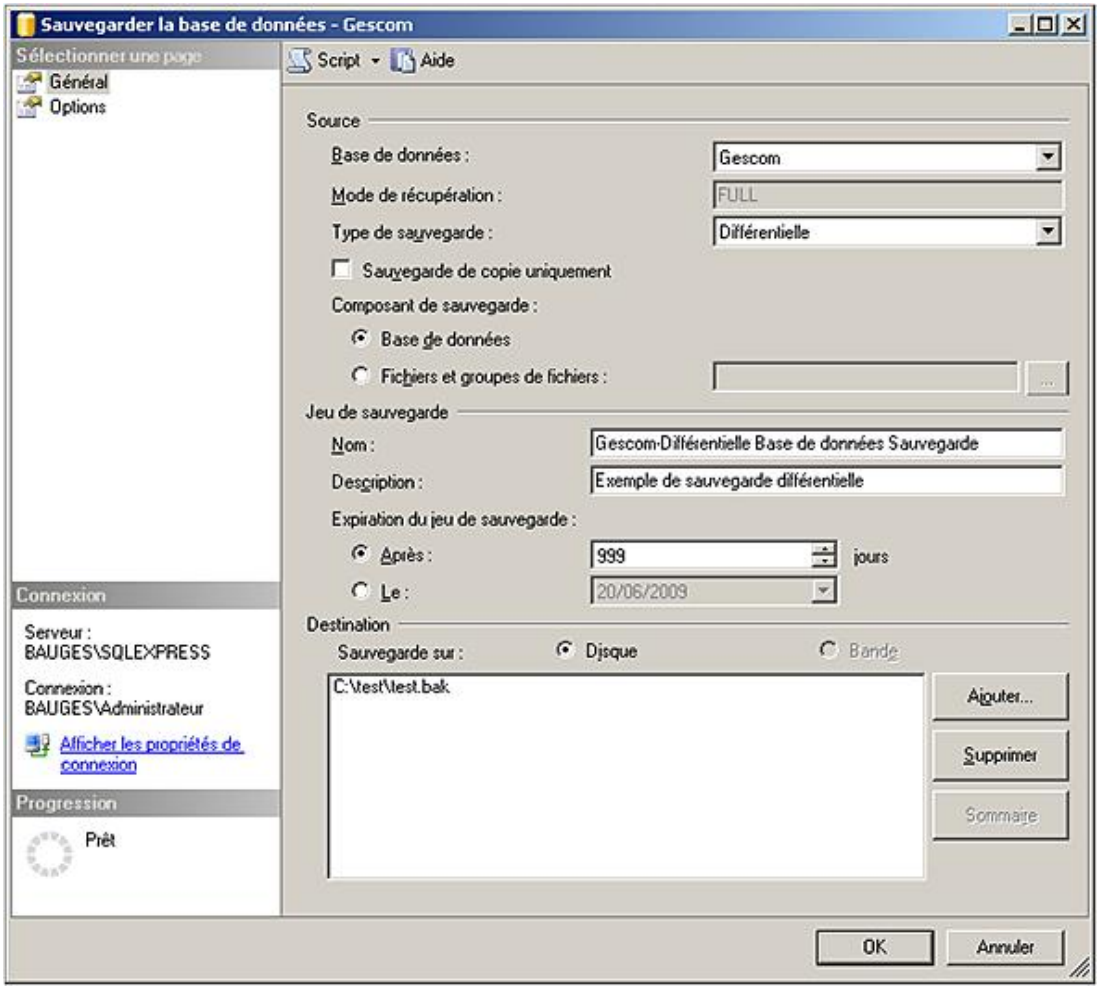



*Sauvegarde complète de Gescom en Transact SQL*

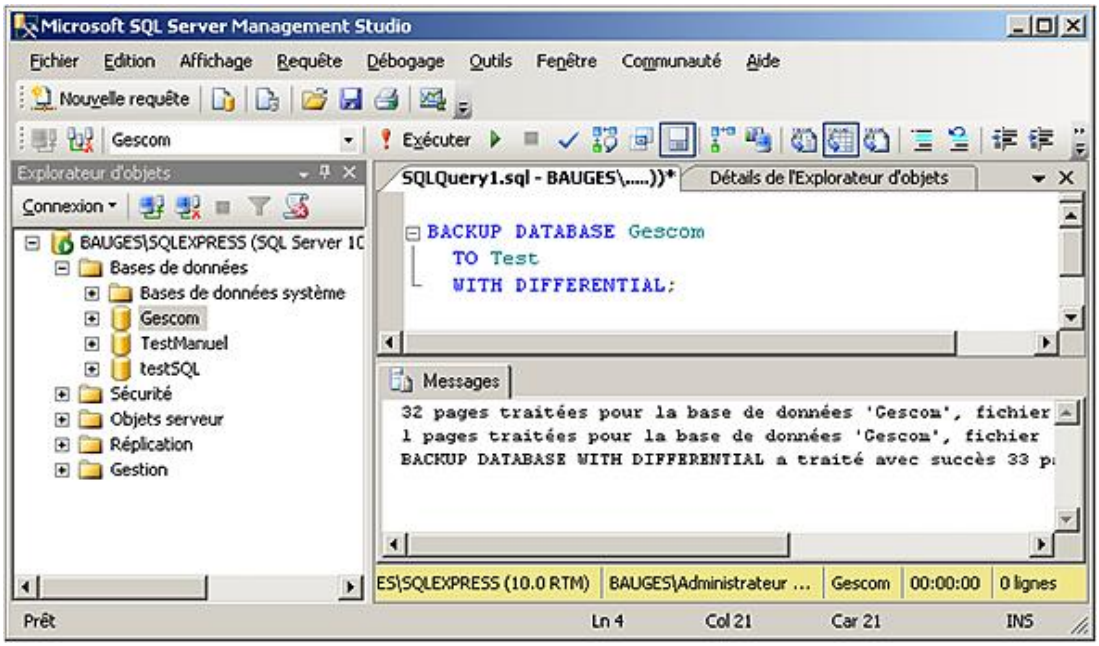
### b. Sauvegarde différentielle

Ce type de sauvegarde n'est possible que si une sauvegarde complète a été préalablement effectuée. Seules les

pages modifiées depuis la dernière sauvegarde complète sont sauvegardées. Pour connaître ces pages, SQL Server utilise le numéro de séquence du journal (LSN : *Log Sequence Number*) de la page qu'il compare avec celui de la synchronisation de la dernière sauvegarde complète.



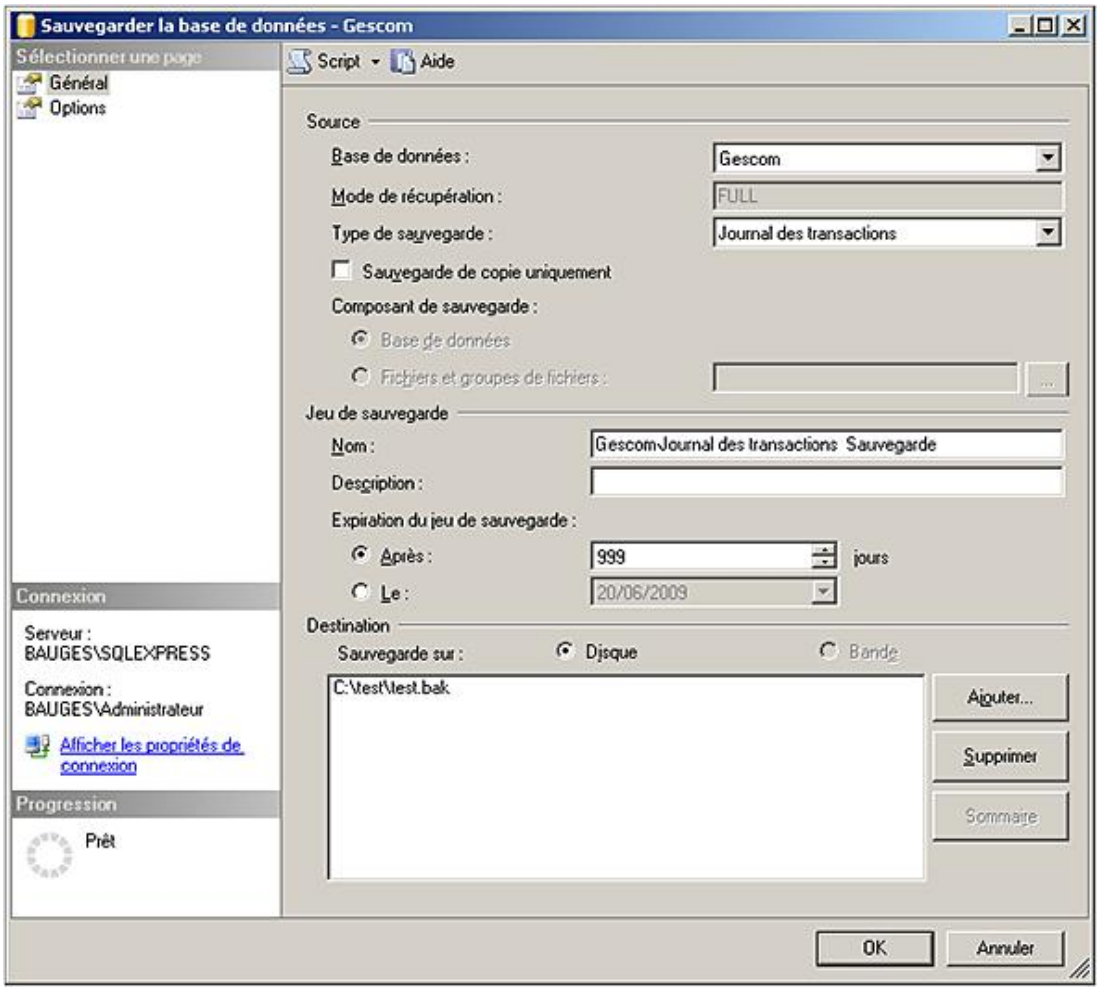

*Sauvegarde différentielle depuis SQL Server Management Studio*



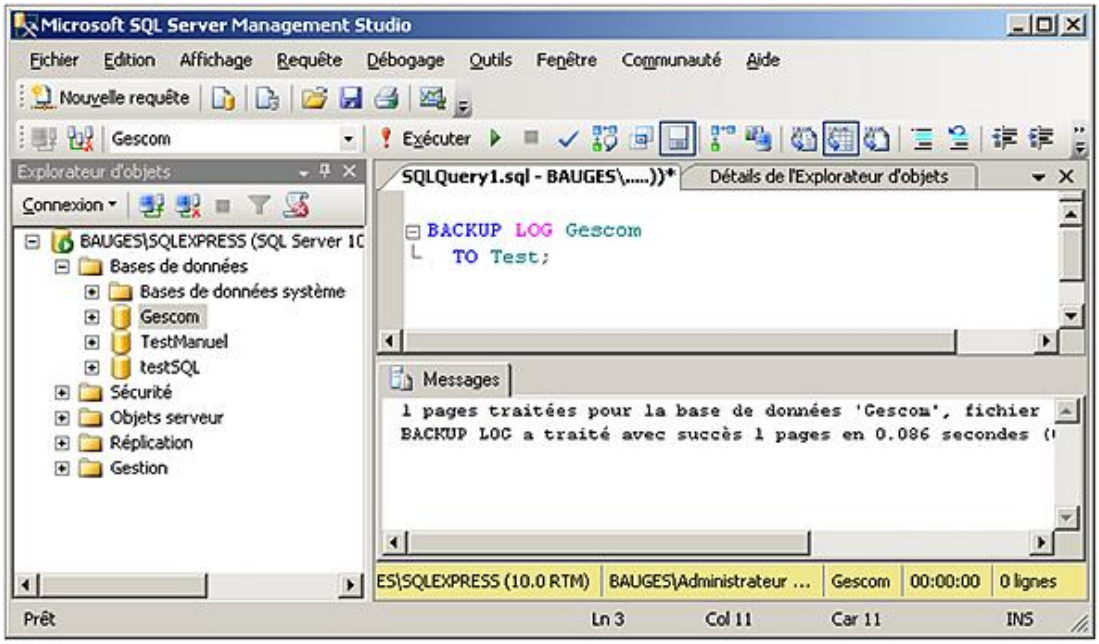

*Sauvegarde différentielle en Transact SQL*

### c. Sauvegarde du journal des transactions

Le journal des transactions peut être sauvegardé. C'est l'instruction BACKUP LOG qui est utilisée. Après sa sauvegarde, la partie inactive du journal est automatiquement tronquée. L'option NO_TRUNCATE permet de ne pas tronquer le journal. Cette option est particulièrement utile dans le cadre d'une opération de restauration.



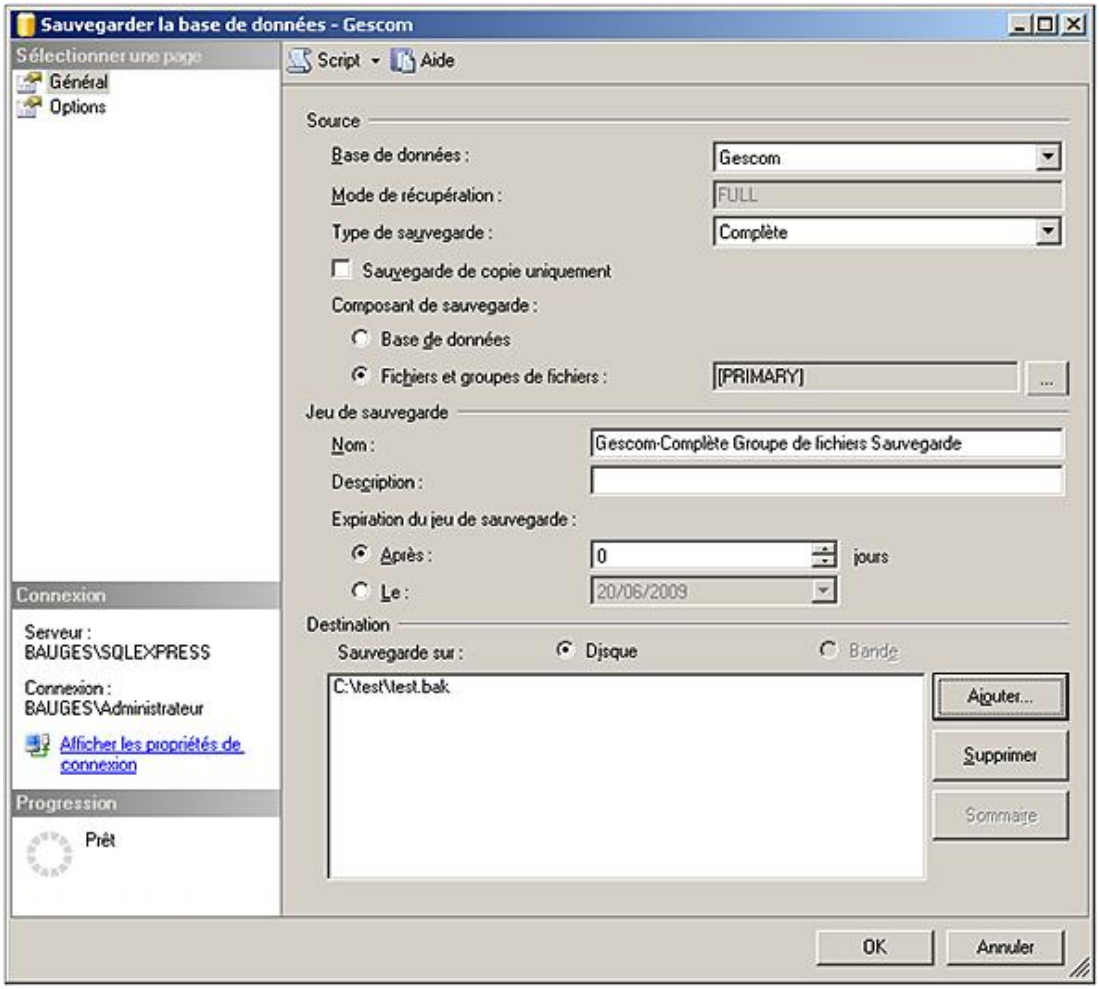

*Sauvegarde du journal des transactions depuis SQL Server Management Studio*




*Sauvegarde du journal des transactions en Transact SQL*

## d. Sauvegarde de fichier ou de groupe de fichiers

Ce type de sauvegarde est particulièrement intéressant pour des bases de très grand volume (VLDB : *Very Large Database*) lorsque les temps de sauvegarde sont longs. Si les objets sont créés sur des groupes de fichiers bien spécialisés, les sauvegardes peuvent être optimales au niveau du temps. Les politiques de sauvegarde possibles sont les mêmes que celles au niveau de la base.




*Demande de sauvegarde d'un fichier ou d'un groupe de fichiers*

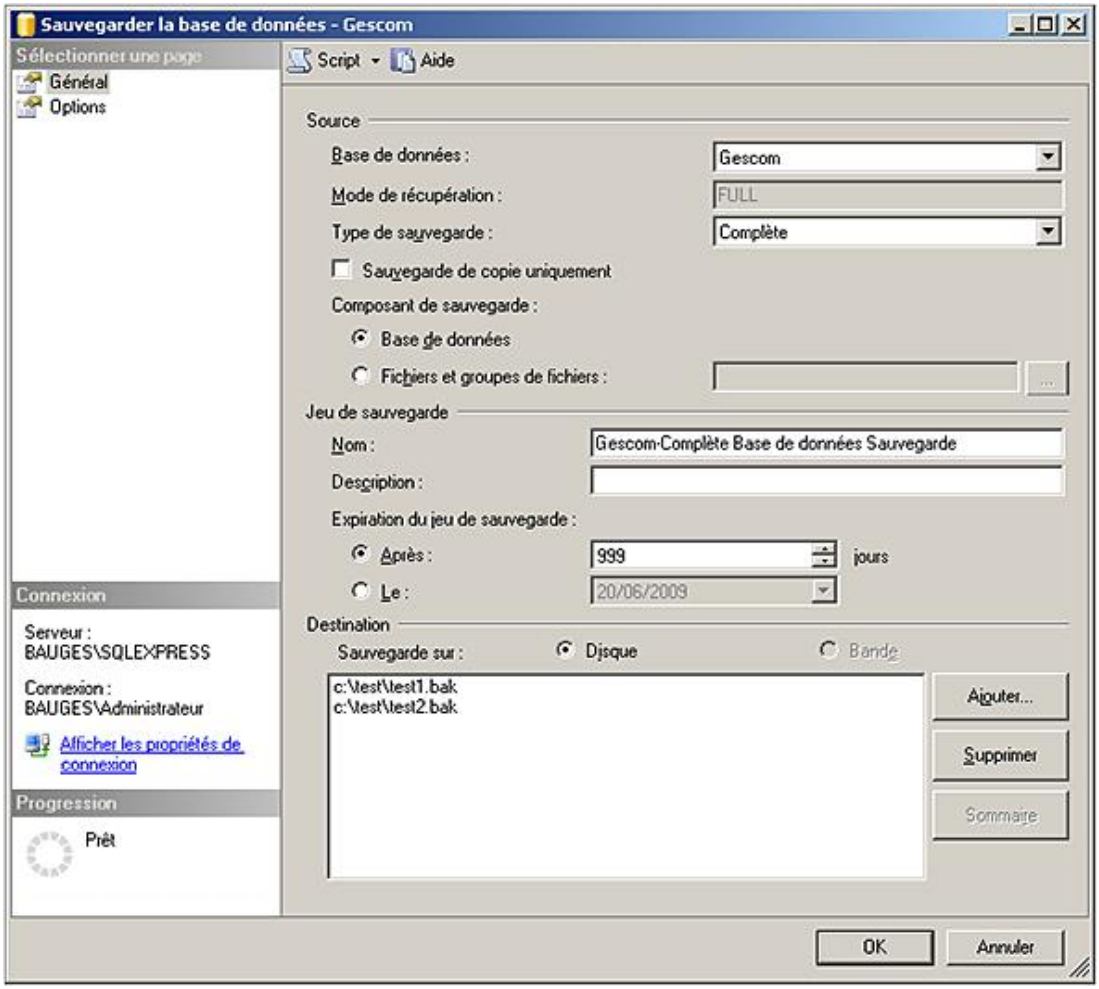
## e. La sauvegarde sur plusieurs fichiers

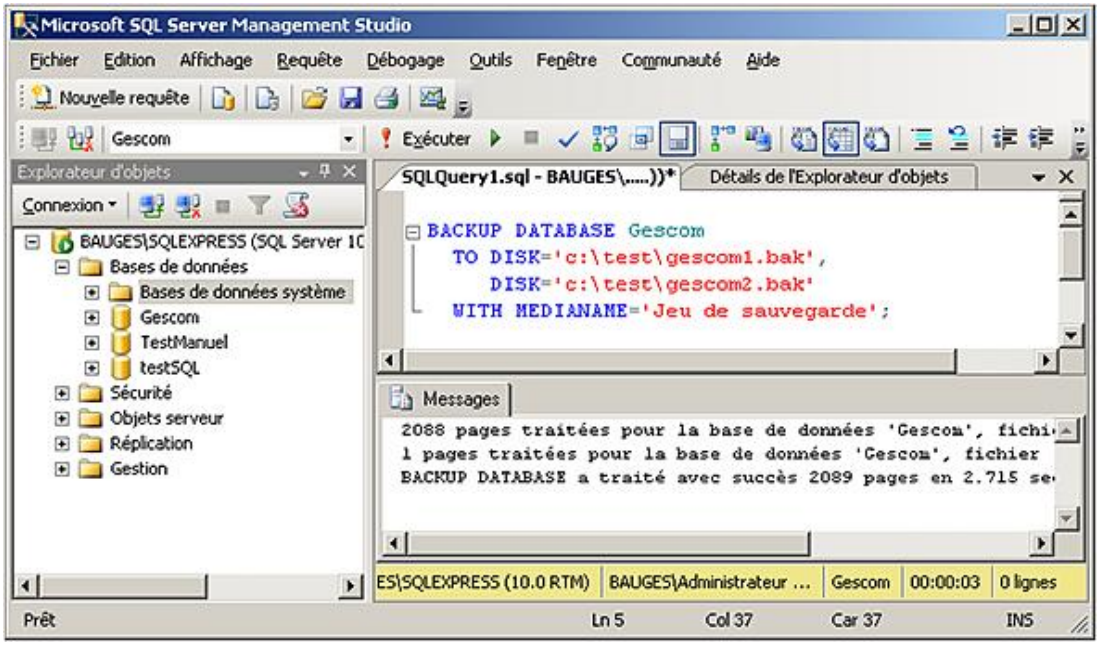
Toujours dans le souci d'accélérer les sauvegardes, il est possible de réaliser des sauvegardes en utilisant plusieurs fichiers simultanément. Les écritures sont effectuées de façon parallèle. C'est l'ensemble des fichiers qui constitue la sauvegarde. Bien que cette méthode permette d'accélérer considérablement les temps, les sauvegardes sont nettement plus fragiles car la perte d'un seul des fichiers empêche l'utilisation de la totalité de la sauvegarde. Les fichiers sont indifféremment des fichiers temporaires ou permanents.

Quelques limites apparaissent :

- tous les fichiers doivent être sur le même type de support (bande, disque),
- si un fichier est membre d'un jeu de sauvegarde, il ne peut l'être que dans le cadre de ce jeu de sauvegarde.

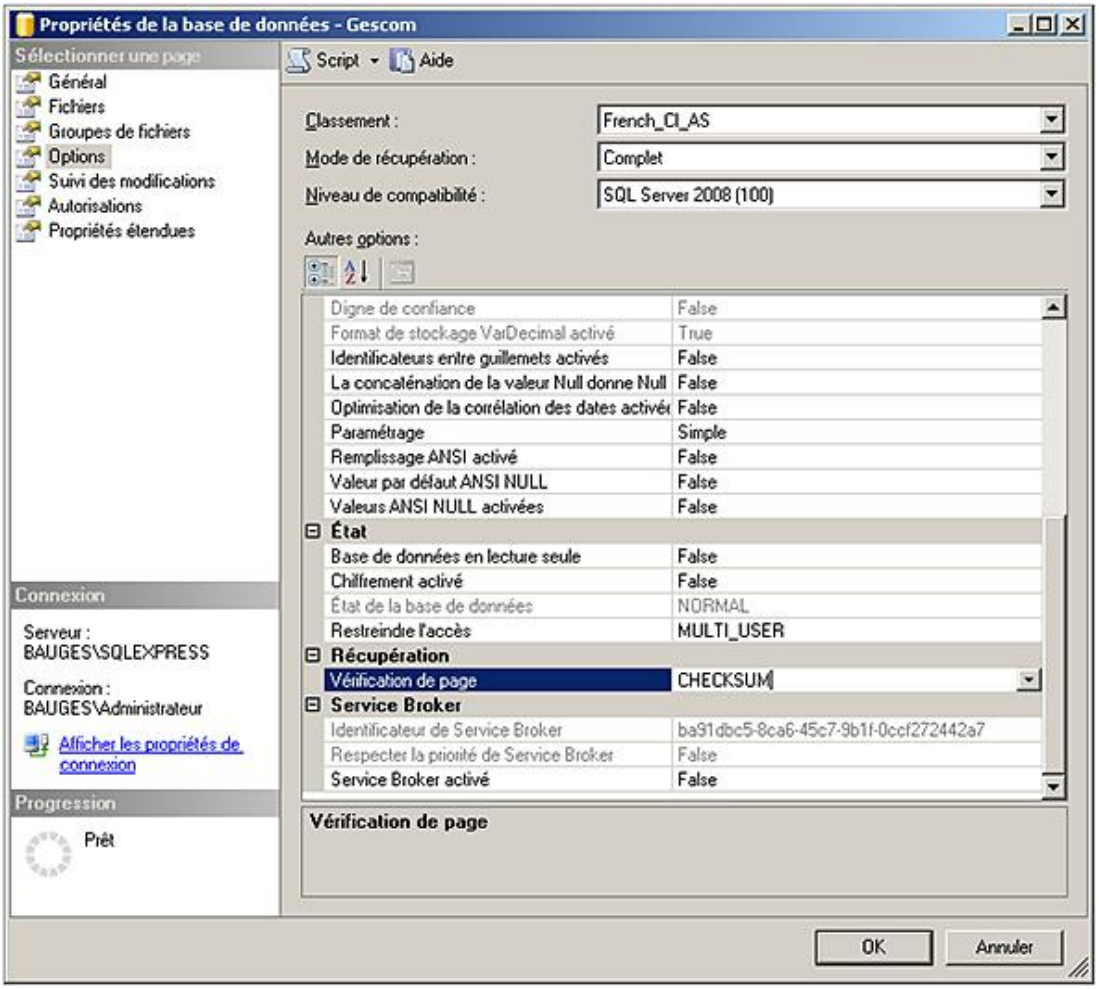
L'instruction BACKUP possède l'option MEDIANAME permettant de nommer un jeu de sauvegarde. Il est ainsi plus aisé de le manipuler. Le nom est limité à 128 caractères.



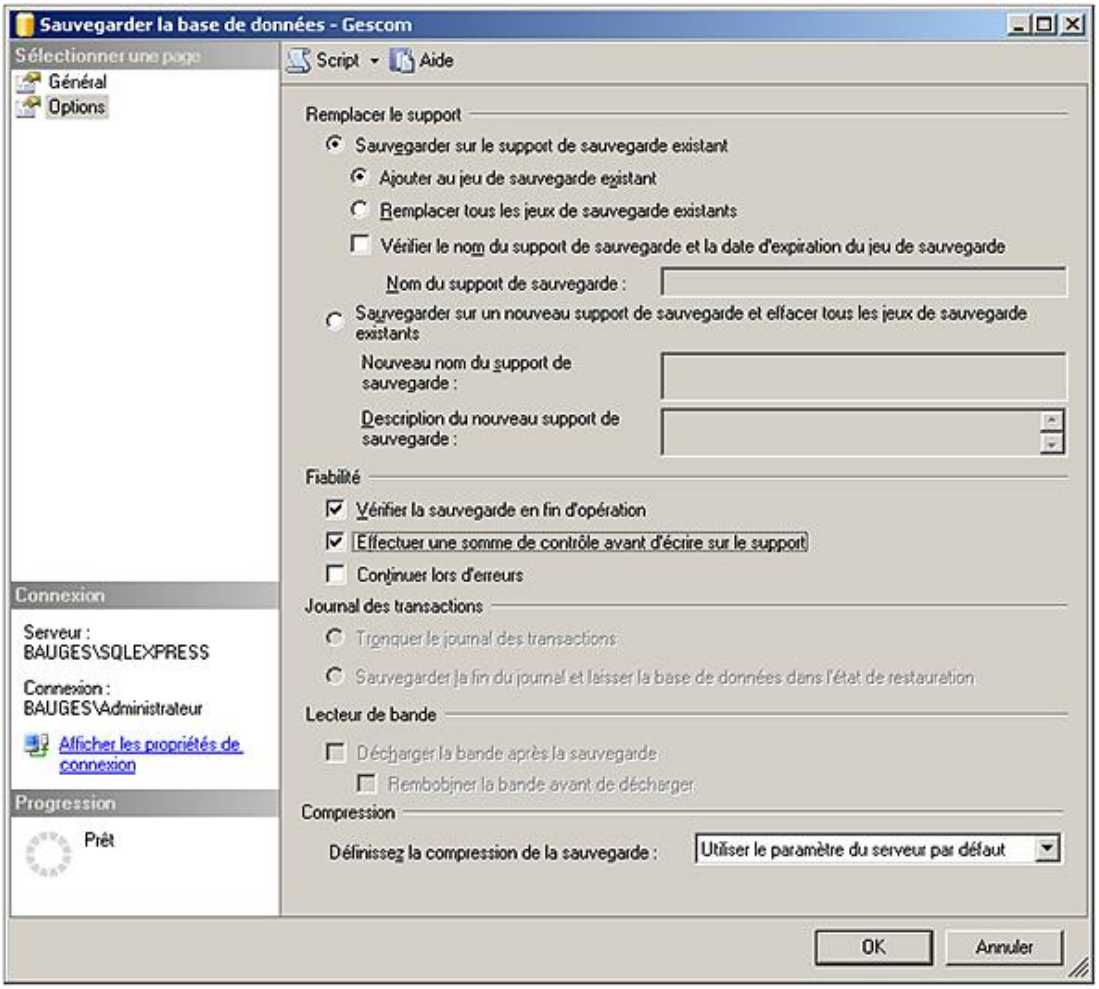




## 5. Vérifier l'intégrité de sauvegarde

Si l'option TORN_PAGE_DETECTION ou CHECKSUM est activée au niveau de la base de données, les sauvegardes réalisées à partir de cette base intègrent la validation de l'intégrité des pages de données dans la sauvegarde. Cette option est définie au niveau de la base.




Lors de la définition d'une sauvegarde, il est possible de demander la vérification de la cohérence de cette sauvegarde depuis la page **Options**.




Il est également possible de vérifier la cohérence du jeu de sauvegardes à l'aide de l'instruction RESTORE VERIFYONLY.

# Vue d'ensemble du processus de restauration

La restauration représente l'opération inverse d'une sauvegarde. Le processus de restauration ne peut pas être utilisé pour réaliser des migrations de base de données.

Cette opération peut être réalisée soit à l'aide des commandes Transact SQL (RESTORE) soit par l'intermédiaire de la console d'administration SQL Server Management Studio. Quelle que soit la base à restaurer, il est indispensable d'installer SQL Server pour remonter les données sur la machine.

Le simple fait de replacer les fichiers constituant les données et le journal ne constitue en aucun cas une restauration, puisque SQL Server n'est pas capable d'accéder à ces fichiers tant qu'ils ne sont pas référencés dans la base Master.

Le processus de restauration peut intervenir dans deux cas :

- suite à une demande explicite de la part d'un utilisateur,
- lors d'un redémarrage du serveur qui fait suite à un arrêt brutal, on parlera alors de restauration automatique.

## 1. La restauration automatique

Ce processus intervient lors de chaque démarrage du serveur. Il s'assure que la dernière opération inscrite dans le journal est un point de synchronisation. Si ce n'est pas le cas, alors le journal est relu depuis le dernier point de synchronisation et toutes les transactions validées sont rejouées tandis que toutes les autres modifications sont annulées. Cette opération est nécessaire afin de garantir la cohérence des données.

## 2. Opérations exécutées automatiquement par SQL Server

SQL Server réalise automatiquement un certain nombre d'opérations afin d'accélérer le processus de restauration et de réduire au maximum le temps d'indisponibilité du serveur.

#### Le contrôle de sécurité

L'intérêt principal de ce contrôle de sécurité est de se prémunir des restaurations accidentelles, qui peuvent écraser une base existante pour remonter une version précédente de la base. De même, le contrôle de sécurité va s'assurer qu'il possède tous les fichiers participant à un jeu de sauvegarde.

Le contrôle de sécurité interdira également la restauration de la base si le jeu des fichiers constituant la base est différent de celui enregistré par le jeu de sauvegarde.

#### Reconstruction de la base de données et des fichiers associés

Dans le cadre d'une restauration à partir d'une sauvegarde complète de la base, SQL Server se charge de recréer la base et les fichiers qui la composent. Les objets sont créés et les données sont transférées. Le schéma de la base de données est donc reconstruit automatiquement durant la procédure de restauration et ne nécessite pas d'opérations manuelles.

## 3. Opérations préliminaires

Avant de réaliser une restauration, il est important de réaliser quelques opérations préliminaires afin de s'assurer du bon déroulement du processus.

### a. La vérification des sauvegardes

Cette opération devrait normalement intervenir à la fin de chaque sauvegarde. Ici, le but est de retrouver la ou les sauvegardes qui vont être nécessaires pour restaurer la base en réduisant au maximum les temps de restauration et le volume des données perdues.

Il existe quatre instructions, qui sont détaillés ci-après mais il est également possible de passer par SQL Server Management Studio.

#### RESTORE HEADERONLY

Permet de connaître les informations contenues dans l'en-tête d'un fichier ou d'un jeu de sauvegarde :

- le nom et la description du fichier ou du jeu de sauvegarde,
- le support utilisé (disque ou bande),
- la date et l'heure de la sauvegarde,
- la méthode de sauvegarde utilisée (complète, différentielle, journal par groupe de fichiers ou complet),
- la taille de la sauvegarde,
- le numéro séquentiel de la sauvegarde dans une chaîne de fichiers de sauvegarde.

#### RESTORE FILELISTONLY

Cette instruction permet d'obtenir des renseignements sur les fichiers de données et journaux constituant la base de données utilisant ce fichier de sauvegarde. Les informations retournées sont :

- les noms logique et physique des fichiers de données et des fichiers journaux,
- le type de chaque fichier (données ou journal),
- le groupe de fichiers auquel le fichier appartient,
- la taille maximale de chaque fichier exprimée en MégaOctet,
- la taille du jeu de sauvegarde exprimée en MégaOctet.

#### RESTORE LABELONLY

Permet d'obtenir quelques informations sur le fichier de sauvegarde.

#### RESTORE VERIFYONLY

Cette instruction, qui ne vérifie pas la cohérence des sauvegardes, n'est utilisée que dans le cadre de jeu de sauvegarde pour s'assurer que tous les fichiers qui constituent ce jeu sont présents.

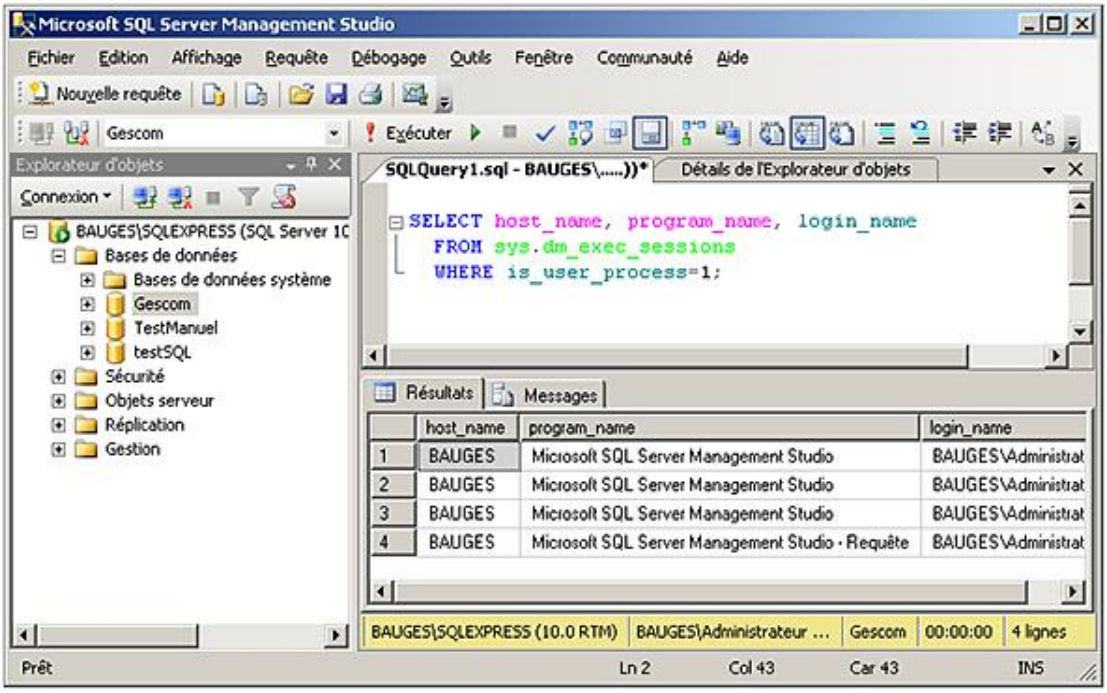
Ces quatre instructions permettent de visualiser le contenu d'un jeu de sauvegarde et fournissent donc beaucoup d'information sur la structure la base de données à restaurer. Seuls les utilisateurs disposant du privilège CREATE DATABASE sont en mesure d'exécuter ces instructions. Ce niveau de privilège est nécessaire à partir de SQL Server 2008.

### b. Les tâches spécifiques

Avant de lancer une opération de restauration sur une base de données, il faut s'assurer, si la base existe déjà que personne ne travaille dessus, puis dans un deuxième temps, sauvegarder la partie du journal des transactions pour laquelle on ne possède pas de sauvegarde afin de minimiser la perte de données.

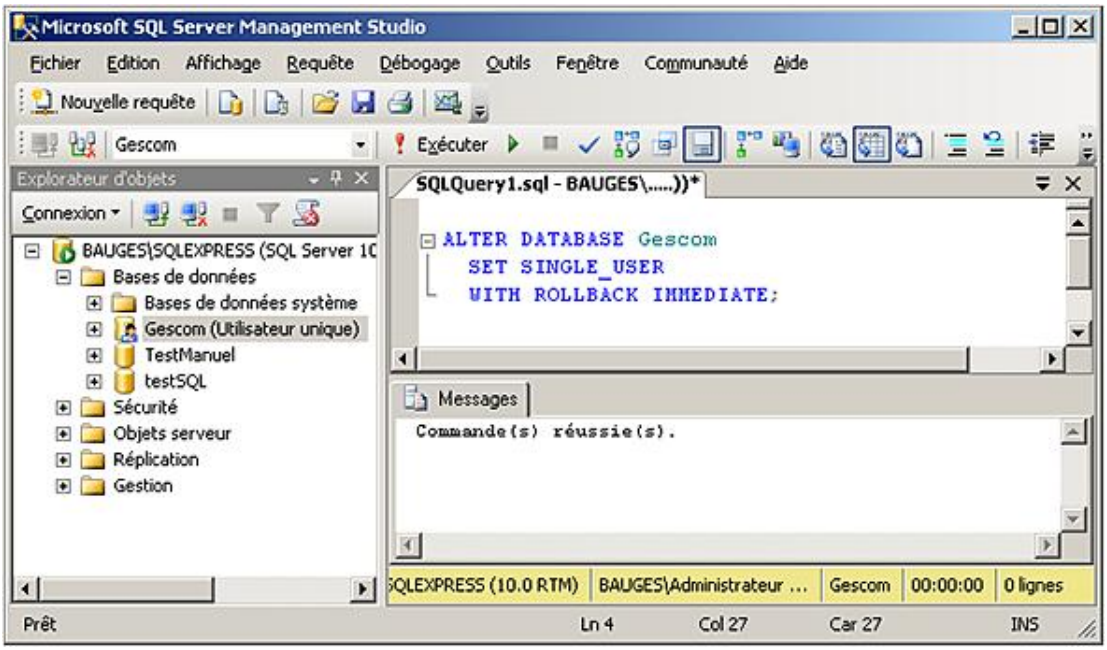
#### S'assurer qu'aucun utilisateur ne travaille sur la base

Il est possible en interrogeant la vue système **sys.dm_exec_sessions** d'établir la liste des utilisateurs actuellement connectés à la base. L'exemple ci-dessous permet de connaître la connexion, le nom du programme et le poste utilisé pour établir les connexions utilisateur (is_user_process=1).



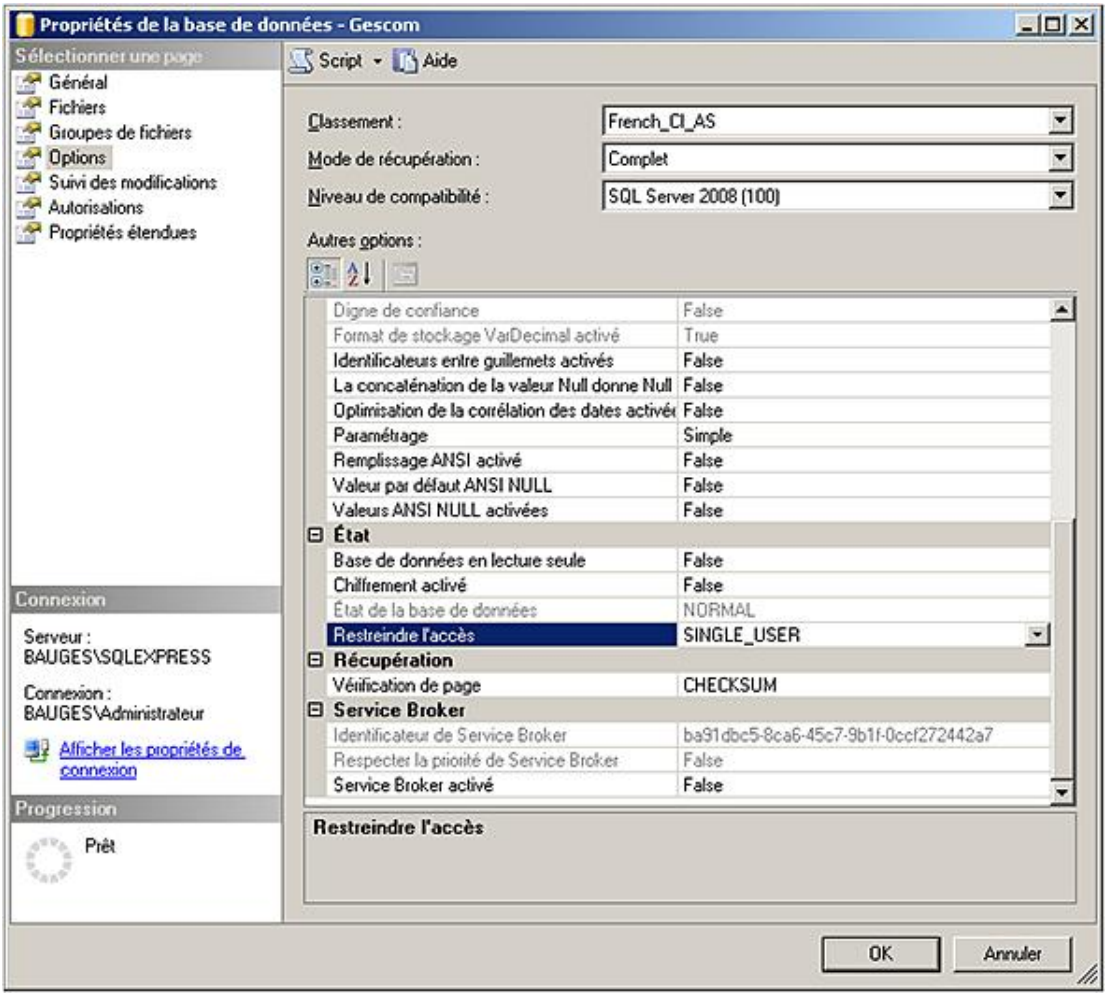

Après s'être assuré qu'il ne reste plus d'utilisateurs connectés à la base, il est nécessaire de garantir le fait que de nouvelles connexions ne peuvent pas être établies en plaçant la base en mode mono-utilisateur.

C'est un administrateur de la base de données qui peut en restreindre l'accès, soit à un seul utilisateur, soit aux seuls utilisateurs qui possèdent le privilège d'ouvrir une session lorsque la base se trouve en mode restreint.




*Changement du mode d'accès en Transact SQL*

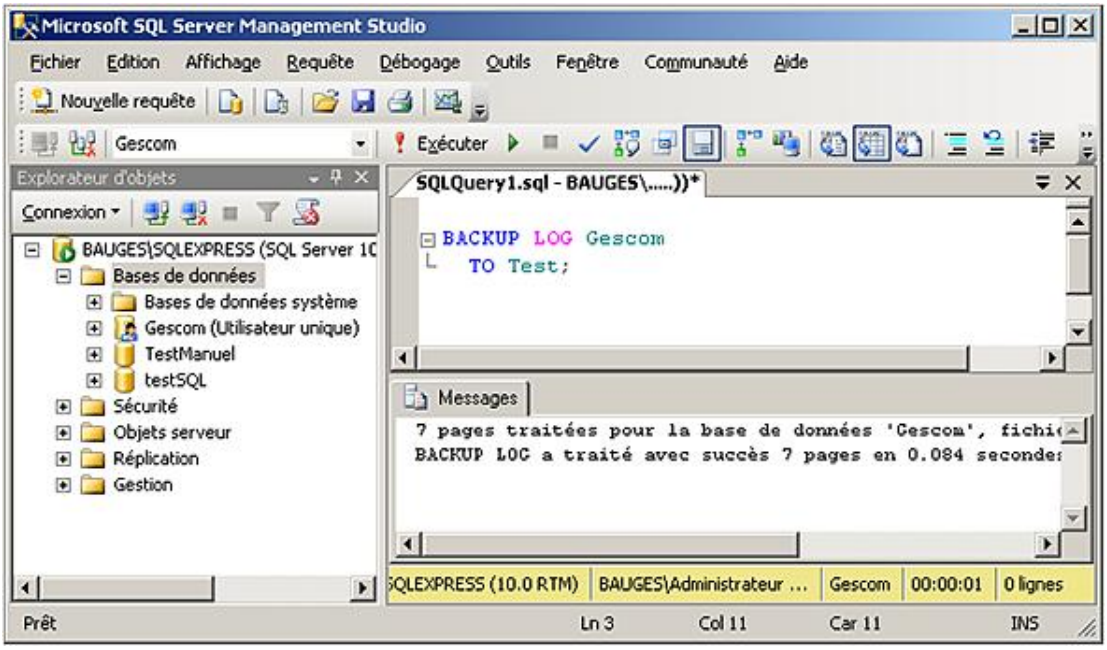
L'instruction ALTER DATABASE GESCOM SET MULTI_USER permet de revenir en mode d'accès normal.




*Changement du mode d'accès depuis SQL Server Management Studio*

## Sauvegarde du journal des transactions

Lorsque cette opération est possible, elle est faite par l'intermédiaire d'une instruction BACKUP LOG.




*Sauvegarde du journal des transactions en cours*

# Restauration des sauvegardes

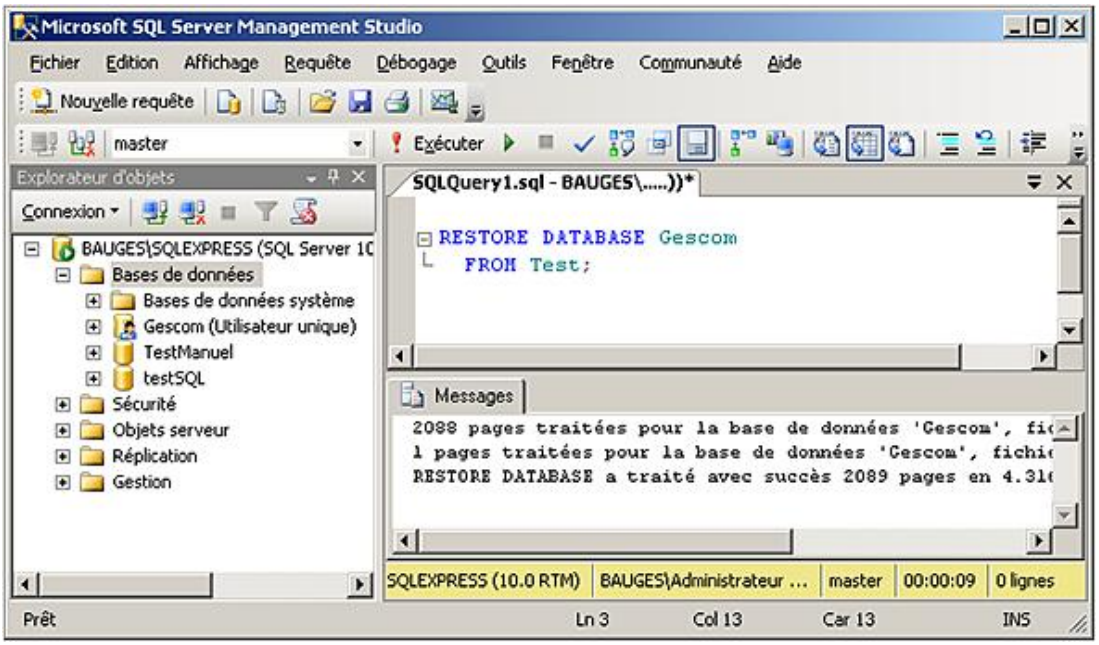
Suivant la sauvegarde effectuée, la méthode de restauration va être légèrement différente.

## 1. L'instruction RESTORE

En mode Transact SQL, c'est l'instruction RESTORE qui permet de remonter une sauvegarde faite par SQL Server.

**Syntaxe**

```
RESTORE DATABASE {nom_base |@var_nom_base }
[FROM unite_de_sauvegarde [,...n ] ]
[WITH
[{CHECKSUM | NO_CHECKSUM }]
[[,] { CONTINUE_AFTER_ERROR | STOP_ON_ERROR } ]
[[,] FILE = numéro_fichier]
[[,] KEEP_REPLICATION]
[[,] MEDIANAME = nom_support]
[[,] MEDIAPASSWORD = mot_de_passe_media]
[[,] MOVE 'nom_logique' TO 'nom_physique'] [,...n]
[[,] PASSWORD = mot_de_passe]
[[,] PARCIAL]
[[,] {RECOVERY|NORECOVERY|STANDBY = nom_fichier_annulation}]
[[,] REPLACE]
[[,] RESTRICTED_USER]
[[,] {REWIND|NOREWIND}]
[[,] STATS [ = pourcentage]]
[[,] {STOPAT = date_heure |
     STOPATMARK = {'marque'|'lsn:numéro_lsn' }
     [ AFTER date_heure] |
     STOPBEFOREMARK = { 'marque' | 'lsn:numéro_lsn' }
     [ AFTER date_heure]}]
[[,] {UNLOAD|NOUNLOAD}]
]
[;]
```




La commande RESTORE doit toujours être lancée depuis la base **Master**.

Pour restaurer le journal des transactions, il est nécessaire d'utiliser l'instruction RESTORE LOG qui possède un paramétrage similaire à celui de RESTORE DATABASE.

Sur une base de données endommagée, la restauration permet de retrouver un ensemble cohérent de données. L'étape de restauration se charge de recréer automatiquement les fichiers et les objets de la base, sans pour autant qu'il soit nécessaire de supprimer la base de données auparavant.

## 2. Les options de l'instruction RESTORE

Il existe de nombreuses options sur l'instruction RESTORE, seules certaines sont détaillées ici.

Tout d'abord lorsque la restauration utilise plusieurs sauvegardes (une complète puis une différentielle et enfin celle des journaux de transactions) il est important que SQL Server ne rende pas la base accessible aux utilisateurs tant que la dernière restauration n'a pas été effectuée. Pour cela, l'instruction RESTORE propose les options RECOVERY et NORECOVERY.

#### RECOVERY

C'est l'option pas défaut qui est utilisée par SQL Server. Avec cette option, à la fin de la restauration, SQL Server passe en revue le journal de transactions afin d'annuler toutes les transactions non validées depuis le dernier point de synchronisation et de confirmer toutes les transactions validées. Une fois ces opérations terminées, la base est accessible par les utilisateurs.

#### NORECOVERY

Cette option doit être spécifiée pour toutes les étapes de la restauration sauf la dernière. SQL Server ne touche pas au journal des transactions et la base de données ne peut pas être utilisée.

#### FILE

Cette option est utilisée uniquement lorsque le fichier de sauvegarde contient plusieurs sauvegardes. Elle permet de préciser le numéro de la sauvegarde que l'on souhaite restaurer.

#### MOVE... TO

Par défaut les fichiers de la base de données sont restaurés au même endroit que celui défini lors de la sauvegarde. Si l'on souhaite préciser un chemin différent il faut utiliser cette option.

#### REPLACE

Cette option permet de restaurer une base en écrasant la base existant préalablement sur le serveur. Cette option est désactivée par défaut.

#### STOPAT

Cette option, disponible uniquement si la base est configurée en mode de restauration complète, permet de rejouer les transactions enregistrées dans le journal jusqu'à une date et heure spécifiées au format varchar, char, smalldatetime ou datetime.

#### STOPATMARK, STOPBEFOREMARK

Ces options sont disponibles uniquement si la base est configurée en mode de restauration complet. Elles permettent de restaurer les transactions jusqu'à une transaction marquée ou bien un numéro d'enregistrement (*Log Sequence Number*).

## 3. La restauration des différents types de sauvegarde

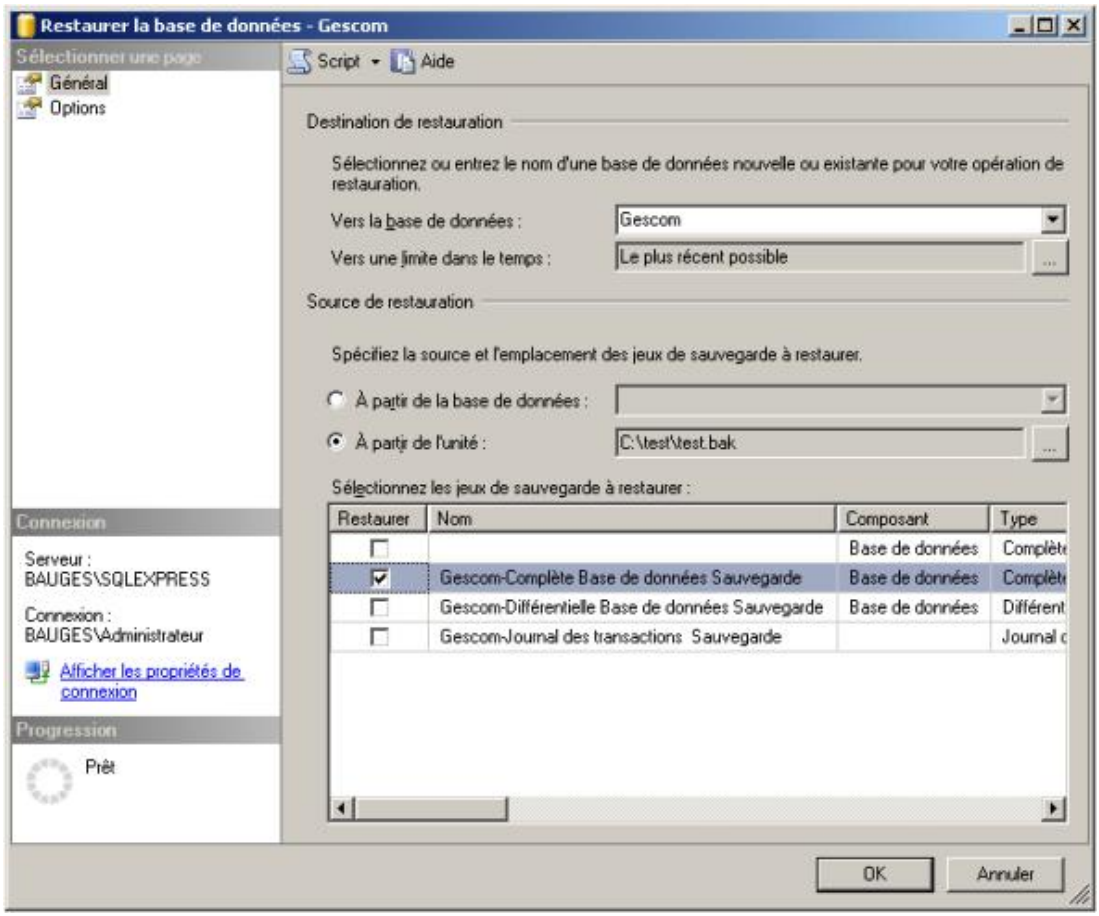
Toutes les politiques de restauration commencent nécessairement par la récupération d'une sauvegarde complète de la base. Une fois ce point de départ établi, il est possible de cumuler les différentes restaurations.

### a. À partir d'une sauvegarde complète

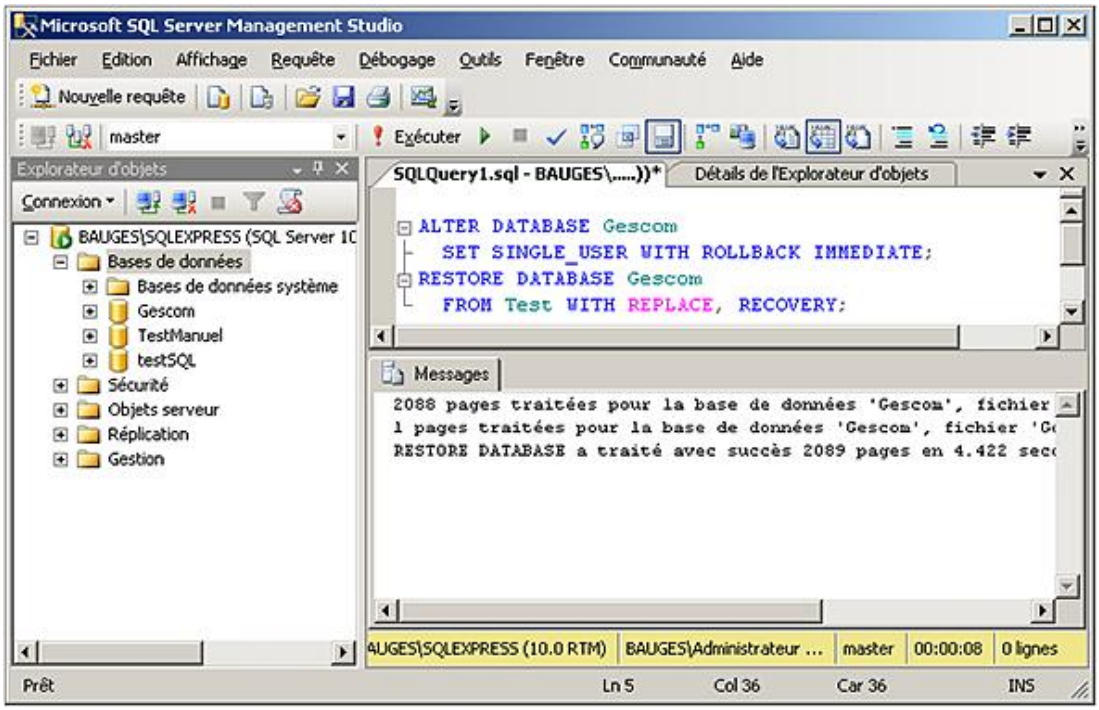
La restauration à partir d'une base complète est une opération simple à réaliser et rapide. S'il est possible de réaliser une sauvegarde complète tous les jours, il faudra le faire car c'est cette méthode qui permet d'obtenir les temps d'indisponibilité du serveur les plus courts.

SQL Server replace tous les fichiers de données et les fichiers associés à leurs emplacements d'origine. De même, l'ensemble du schéma de la base est récupéré automatiquement.

Une telle opération de restauration est réalisée lorsque le disque physique contenant les fichiers de la base de données est endommagé, ou bien lorsqu'une partie des fichiers est corrompue ou perdue. La restauration d'une sauvegarde complète est également envisageable lorsque l'on souhaite restaurer les données sur un deuxième serveur (ce serveur peut devenir un serveur de secours par exemple).




*Restauration depuis SQL Server Management Studio*




*Restauration sans mise en ligne de la base*

Les options de récupération RECOVERY ou NORECOVERY devront être précisées surtout si d'autres restaurations suivent.

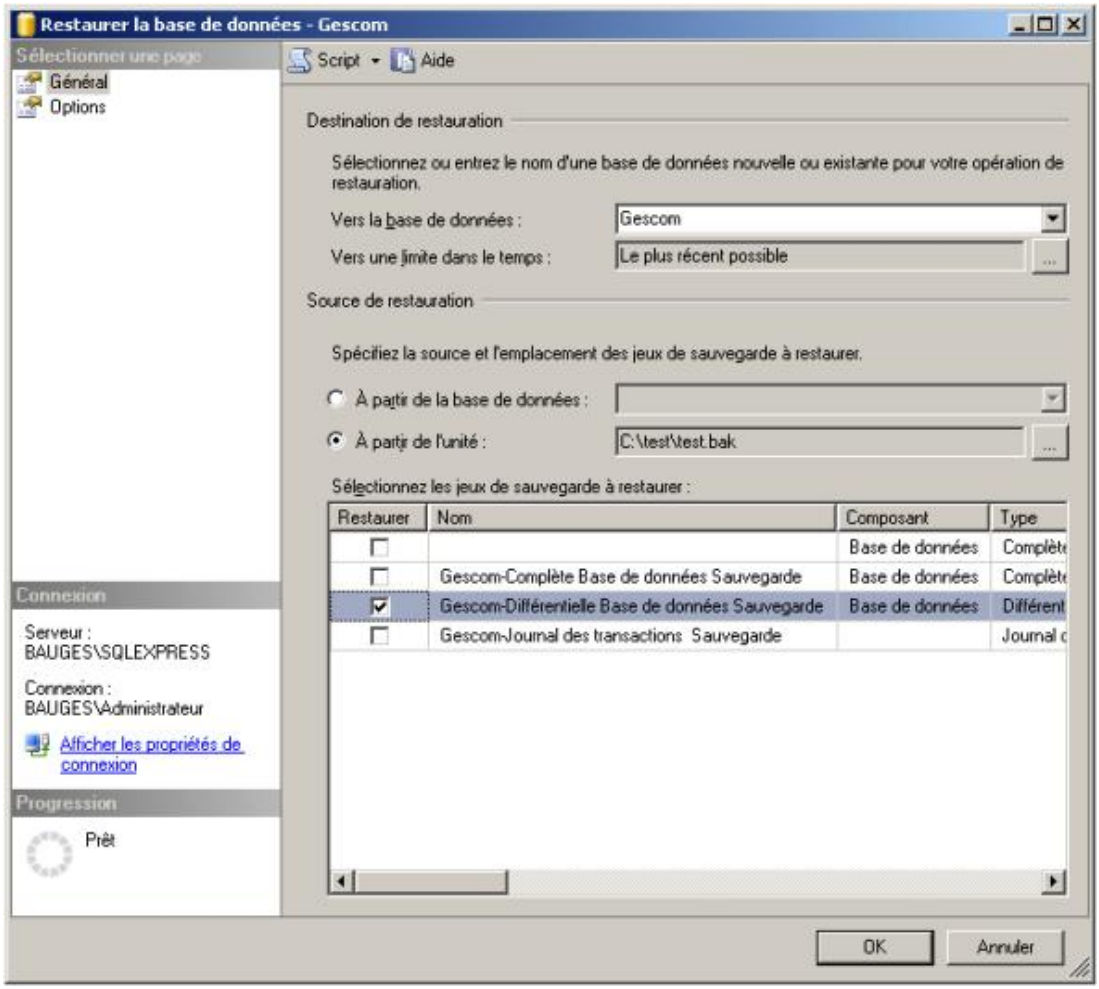
### b. À partir d'une sauvegarde différentielle

Ce type de restauration ne peut intervenir que suite à celle d'une sauvegarde complète.

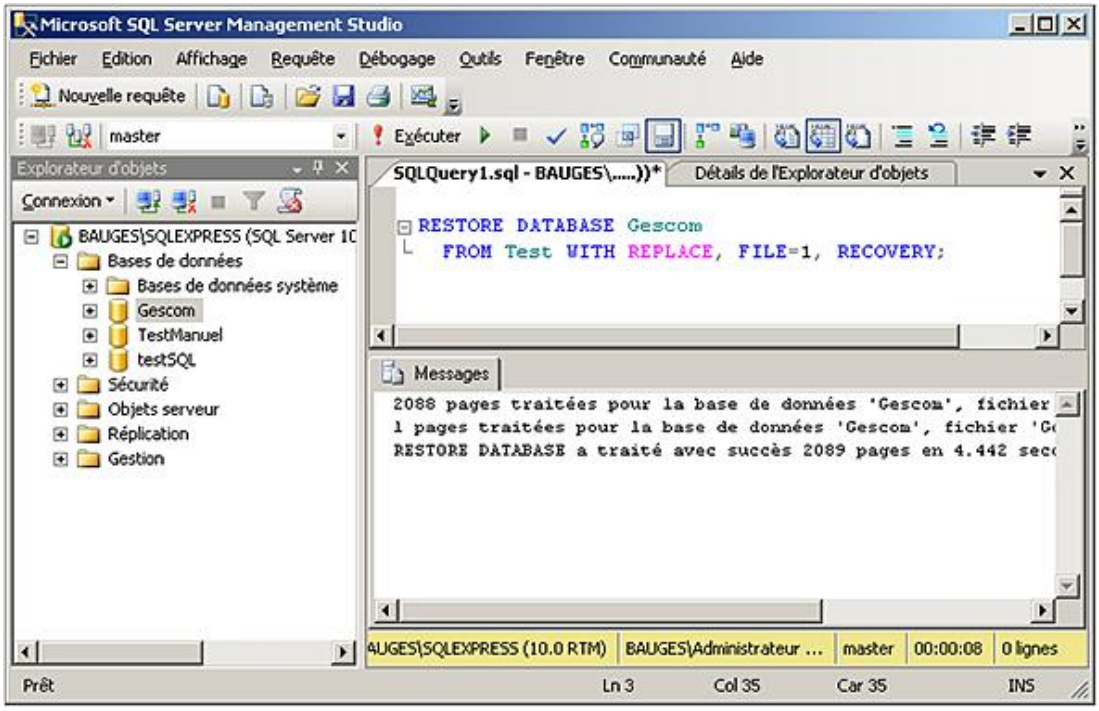
Si plusieurs sauvegardes différentielles existent pour la même base de données, il suffit de restaurer la plus récente car une sauvegarde différentielle contient toutes les modifications intervenues sur les données depuis la dernière sauvegarde complète.

La restauration ramène la base de données à l'état dans lequel elle était lorsque la sauvegarde a été effectuée.

La restauration d'une telle sauvegarde est bien sûr beaucoup plus rapide que celle de tous les journaux de transactions.




*Restauration d'une sauvegarde différentielle*




*Restauration différentielle à partir d'une unité de sauvegarde. On utilise ici la première sauvegarde réalisée dans cette unité.*

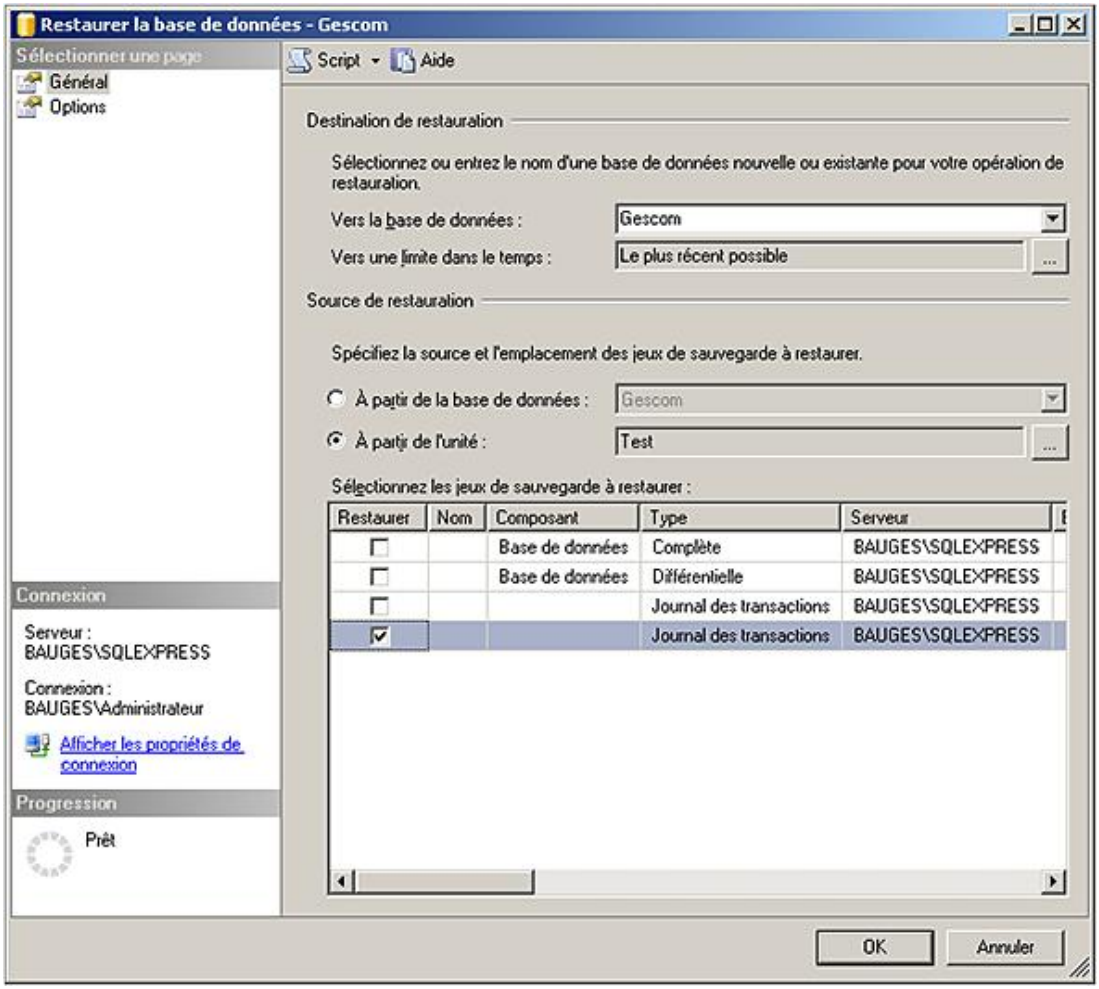
Encore une fois, si la sauvegarde différentielle est la dernière dont on dispose pour la base de données, il faudra exécuter l'ordre RESTORE avec RECOVERY. Dans tous les autres cas (on dispose de sauvegardes du journal des transactions), il faut utiliser l'option NORECOVERY pour être à même de restaurer la base au mieux.

### c. À partir d'une sauvegarde du journal des transactions

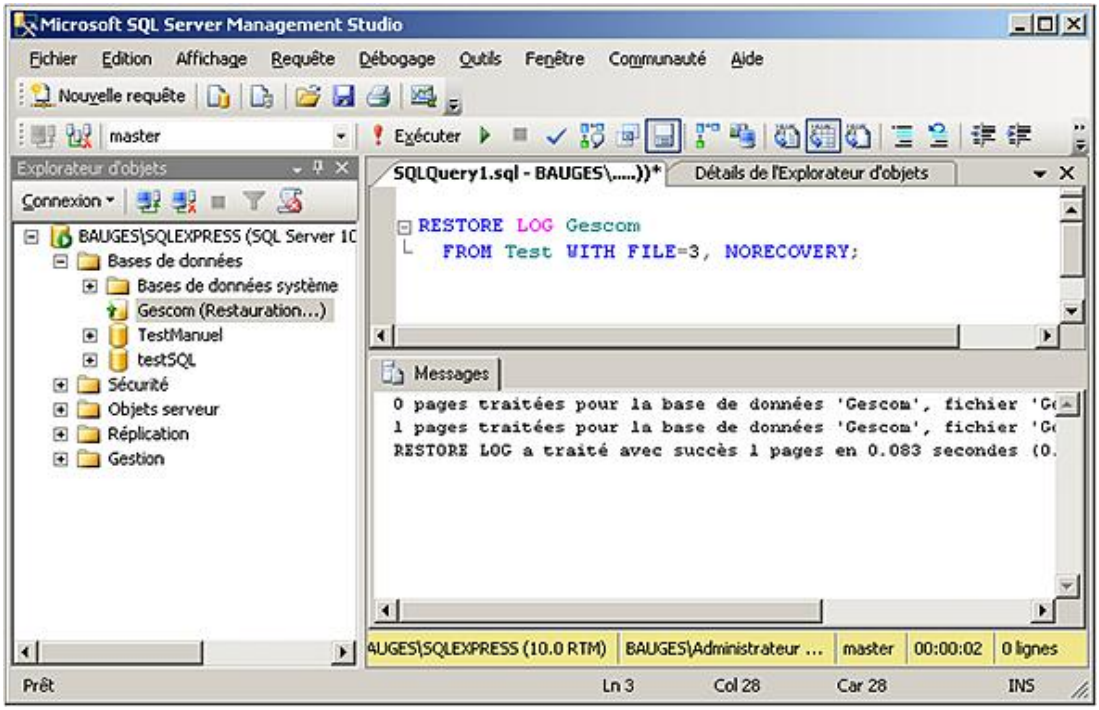
Le journal des transactions est le dernier élément à restaurer. Il va permettre de limiter la perte des informations modifiées dans la base depuis la dernière sauvegarde complète ou différentielle.

Il peut éventuellement y avoir plusieurs fichiers journaux à restaurer, dans ce cas ils doivent l'être dans l'ordre chronologique.

Tous les ordres de restauration des journaux doivent comprendre l'option NORECOVERY, à l'exception de celle du dernier journal où l'option sera soit omise, soit RECOVERY.



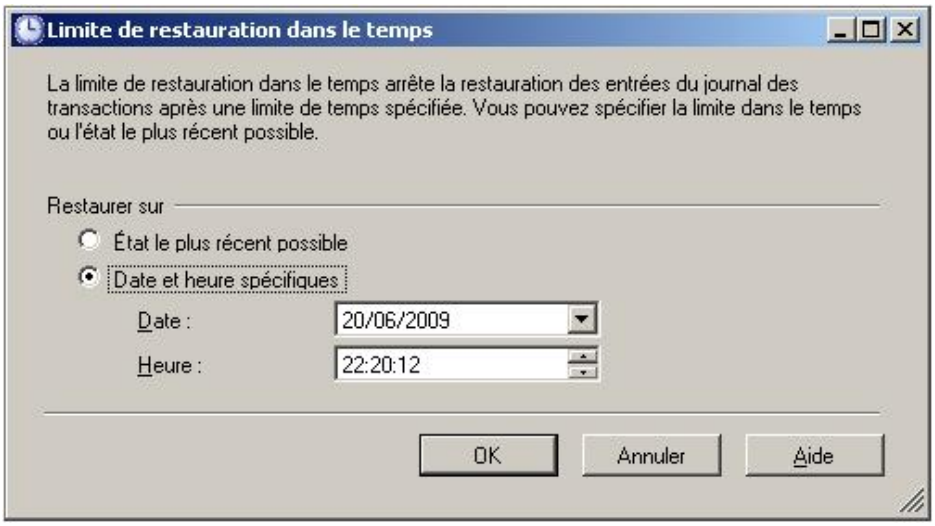

*Demande de restauration du journal depuis SQL Server Management Studio*

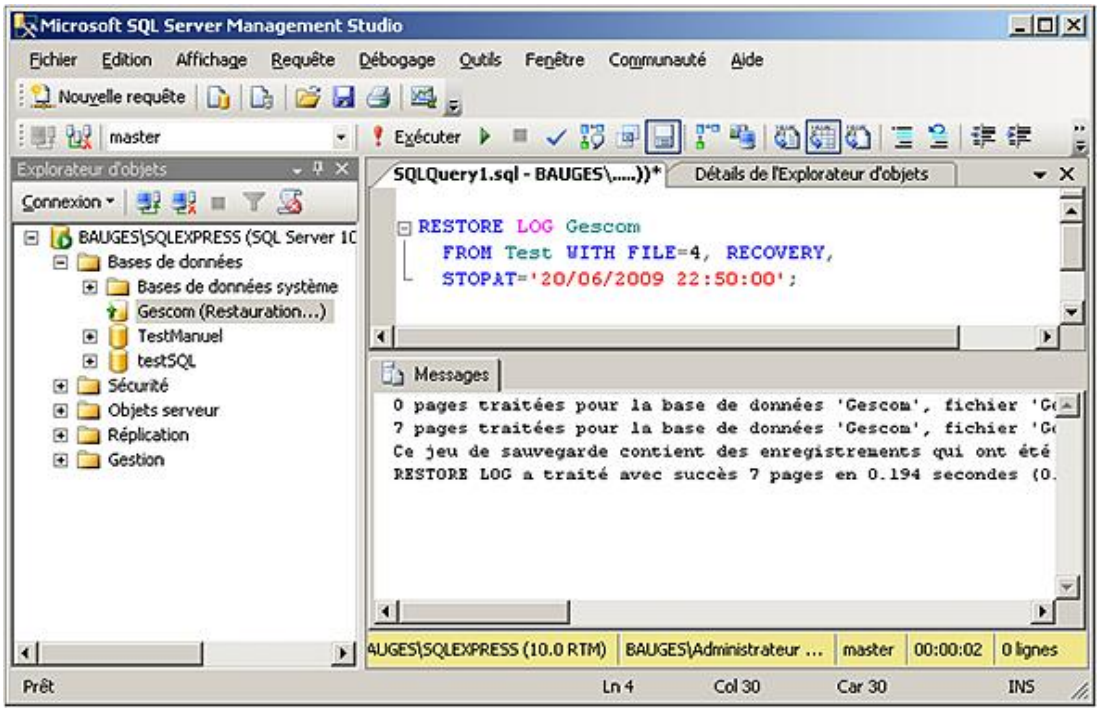



Lors de la restauration des journaux, il est possible de rejouer les transactions jusqu'à une date et heure bien précises. Pour bénéficier de cette fonctionnalité il faut utiliser l'option STOPAT en association avec l'option RECOVERY.

Cette fonctionnalité permet de restaurer une base en s'arrêtant à un instant bien précis et donc de ne pas rejouer des transactions désastreuses.

Depuis SQL Server Management Studio, il est possible lors de la restauration d'un journal de spécifier la limite dans le temps depuis la fenêtre **Limite de restauration dans le temps**.




### d. À partir d'une sauvegarde de fichier ou d'un groupe de fichiers

Les opérations à mener et leur ordre sont les mêmes que dans le cadre d'une restauration totale de la base de données. L'instruction RESTORE se complète après le nom de la base avec le nom du groupe de fichiers qui est concerné par la restauration.

## 4. Restauration des bases de données système endommagées

Suivant la gravité des événement qui sont survenus, plusieurs possibilités sont envisageables.

### a. Restauration à partir d'une sauvegarde

Si le serveur peut être démarré, il est possible de restaurer les bases de données système à l'aide d'une commande RESTORE DATABASE.

### b. Reconstruction de bases de données système

La reconstruction des bases de données système est possible en lançant de nouveau le processus d'installation de l'instance SQL Server.

La reconstruction des bases système écrase les bases **master** et **model** existantes.

Lorsque les bases système sont reconstruites, il est alors possible de redémarrer les services et de procéder à la restauration depuis une sauvegarde valide.

# Ressource sur le Web

| Adresse | Description |
|---|---|
| http://www.apsql.com | Informations diverses sur SQL Server |
| http://www.guss.fr | Groupe des utilisateurs de SQL Server |
| http://www.microsoft.com/france/sql | Site Web Français de SQL Server |
| http://www.microsoft.com/sql | Site Web de SQL Server |
| http://msdn.microsoft.com | MSDN |
| http://msdn.microsoft.com/fr-fr/SQL | Centre de ressources SQL Server sur MSDN |
| http://www.microsoft.com/fr-fr/default.aspx | Technet |
| http://microsoft.com/express/sql/default.aspx | SQL Express |
| http://support.microsoft.com | Support technique de Microsoft |

# Glossaire

**ACL** : Access Control List

**ADF** : Application Definition File

**ADO** : ActiveX Data Object

**API** : Application Programming Interface

**AWE** : Address Windowing Extensions

**B2B** : Business To Business

**BCM** : Bulk Change Map

**BLOB** : Binary Large Object

**CLR** : Common Language Runtime

**COM** : Component Object Model

**CTE** : Common Table Expression

**DCM** : Differential Change Map

**DDL** : Data Definition Language

**DEK** : Database Encryption Key

**DML** : Data Manipulation Language

**DMV** : Dynamic Management View

**DTA** : Database Tuning Advisor

**DSS** : Decision Support System

**ETL** : Extraction Transformation and Load

**GC** : Garbage Collector

**GAC** : Global Assembly Cache

**GAM** : Global Allocation Map

**IAM** : Index Allocation Map

**ICF** : Instance Configuration File

**LCID** : LocaleID

**MARS** : Multiple Active Result Set

**MDAC** : Microsoft Data Access Component

**ODS** : Open Data Services

**OLTP** : On Line Transaction Processing

**OLAP** : On Line Analytical Processing

**PFS** : Page Free Space

**RID** : Row Identifier

**RMO** : Replication Management Object

**SCC** : Setup Consistency Checker

**SGAM** : Shared Global Allocation Map

**SMO** : SQL Management Objects

**SMTP** : Simple Mail Transfer Protocol

**SOA** : Service Oriented Architecture

**SOAP** : Simple Object Access Protocol

**SQL** : Structure Query Language

**TDE** : Transparent Data Encryption

**TSQL** : Transact SQL

**TVF** : Table Valued Functions

**UDF** : User Defined aggregate Function

**UDT** : User Defined Types

**WMI** : Windows Management Instrumentation

**XML** : eXtensible Markup Language